## The SPEC-RG Reference Architecture for the Edge Continuum

Jansen, Matthijs; Al-Dulaimy, Auday; Papadopoulos, Alessandro V.; Trivedi, Animesh; Iosup, Alexandru

2022

# The SPEC-RG Reference Architecture for The Edge Continuum

Matthijs Jansen
*Vrije Universiteit Amsterdam*
Amsterdam, The Netherlands
m.s.jansen@vu.nl

Auday Al-Dulaimy
*Mälardalen University*
Västerås, Sweden
auday.aldulaimy@mdh.se

Alessandro V. Papadopoulos
*Mälardalen University*
Västerås, Sweden
alessandro.papadopoulos@mdh.se

Animesh Trivedi
*Vrije Universiteit Amsterdam*
Amsterdam, The Netherlands
a.trivedi@vu.nl

Alexandru Iosup
*Vrije Universiteit Amsterdam*
Amsterdam, The Netherlands
a.iosup@vu.nl

***Abstract*—Edge computing promises lower processing latencies and better privacy control than cloud computing for task offloading as edge devices are positioned closer to users. Realizing this promise depends on building strong theoretical and engineering foundations of computing based on an *edge continuum* connecting edge to other resources. In the SPEC-RG Cloud Group, we conducted a systematic study of computing models for task offloading and found that these models have many shared characteristics. Despite these commonalities, no systematic model or architecture for task offloading currently exists. In this paper, we address this need by proposing a reference architecture for task offloading in the edge continuum and synthesize its components using well-understood abstractions, services, and resources from cloud computing. We provide domain-specific architectures for deep learning and industrial IoT and show how this unified computing model opens up application development as developers are no longer limited to the single set of constraints posed by current isolated computing models. Additionally, we demonstrate the utility of the architecture by designing a deployment and benchmarking framework for edge continuum applications and investigate the performance of various edge continuum deployments. The framework allows for fine-grained discovery of the edge continuum deployment space, including the emulation of complex networks, all with minimal user input required. To enhance the performance analysis capabilities of the benchmark, we introduce an analytical first-order performance model that can be used to explore multiple application deployment scenarios such as local processing on endpoints or offloading between cloud or edge. The deployment and benchmarking framework is open-sourced and available at https://github.com/atlarge-research/continuum.**

***Index Terms*—Edge Continuum, Reference Architecture, Edge Computing, Resource Management, Task Offloading, Benchmark**

## I. Introduction

Cloud computing represents the de facto standard for computing today, where a user can summon a large fleet of servers and deploy a variety of user-customized infrastructure services (storage, resource management, scaling, monitoring) on them in a few clicks. Many cloud-associated programming concepts, abstractions, and toolchains are well established and understood [1]. In contrast to cloud computing, edge computing is an emerging computing paradigm where most data is generated and processed in the field using decentralized, heterogeneous, and mobile computing devices and servers, often with limited resources. Examples of edge deployments can be found in IoT deployments [2], smart farming [3], smart industry [4], mobile gaming [5], machine learning [6], and self-driving vehicles [7].

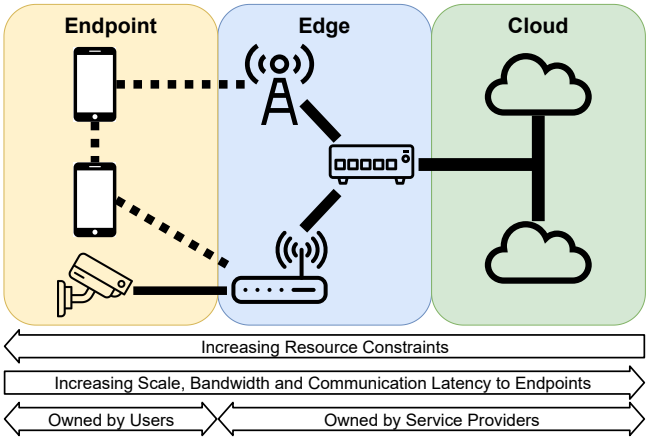


Fig. 1: Overview of the edge continuum.

The key attractive property of edge computing is the possibility to process data in-field, close to the users and source of data, thus offering lower data processing latencies, better user experience, lower cost, and more secure/private data computation than cloud-based computing. These properties are achieved by offloading (either in parts or as a whole) user applications traditionally run in the cloud to edge devices [8]. However, the notion of edge computing has been used in multiple contexts, including but not limited to offloading from cloud to edge [9], offloading from edge to cloud [10], compute management among different edge devices [11], and compute management between edge and mobile devices [8]. Typically, all these deployments have their associated computing models, user services, programming APIs, and abstractions, thus creating complexity and a general confusion regarding *what is edge computing* and *what should a developer or infrastructure provider know about it before developing or supporting edge-ready applications*.

**Key insight:** Existing computing models for task offloading are often presented in isolation, while there is significant overlap in concerns addressed by these models. By considering cloud, edge, and endpoint computing models as part of a unified, continuous computing model—the edge continuum—developers are no longer limited to the single set of constraints posed by current isolated computing models.

In this paper, we take a step back and systematically study the field of computing models for task offloading. More specifically, we study five prevalent models leveraging a combination of cloud, edge, and endpoint resources: mist computing [12], edge computing [13], multi-access edge computing [14], fog computing [15], and mobile cloud computing [16]. We argue that isolated computing models restrict a developer's view of deployment and optimization opportunities outside a specific model silo. Our analysis of these computing models reveals the *primary insight* that even though these computing models are often presented in isolation, there is a significant overlap in terms of concerns addressed, used mechanisms, and application domains addressed by these different computing models. As a result, we make a case that developers and infrastructure providers should not consider cloud, edge, and endpoint computing models in isolation but as parts of a unified, continuous computing model—the *edge continuum* [17]. This edge continuum opens up future advancements as its unified architecture is not limited to a single set of constraints posed by current isolated computing models. Figure 1 provides an overview of the edge continuum.

Based on this insight, we synthesize and design a unified reference architecture for the edge continuum, which identifies the key building blocks of any edge continuum application and associated key concerns. We then map the computing models and related use cases to this reference architecture to demonstrate its completeness, comprehensiveness, and usefulness. Finally, we show what users, developers, infrastructure providers, and cloud service providers should know about the edge continuum to deploy, test, and benchmark applications. The work leading to this reference architecture has been conducted in the SPEC-RG Cloud group.[1]

We present the design and implementation of a deployment and benchmarking framework to investigate the performance of each component of the architecture. The framework focuses on design space exploration and allows users to create various edge continuum deployments in only a few lines of code. We create example deployments for existing edge continuum computing models using many emulated devices distributed over multiple physical machines and demonstrate how our framework can be used to investigate performance differences between these models. Additionally, we create a first-order analytical performance model to predict if application deployments can be processed locally on endpoints or should be offloaded to either cloud or edge.

Our key contributions in this paper include:

1) To the best of our knowledge, we present the most comprehensive survey on computing models for task offloading (Section II), including 16 different models, and perform an analysis of their characteristics, showing significant overlap between these models.
2) We make a case for the edge continuum and propose to design a unified reference architecture for task offloading computing models (Section III). Our reference architecture is the first to consider the *entire edge continuum*, encompassing computing models such as mist computing, (multi-access) edge computing, fog computing, and mobile cloud computing. We use this newly proposed architecture to synthesize two domain-specific architectures for deep learning and industrial IoT.
3) To explore various deployment scenarios across the continuum, we design and implement a deployment and benchmarking framework for the edge continuum (Section IV). We show how users can alter deployments in a few lines of code to examine various design trade-offs.
4) We enhance the performance analysis capabilities of the benchmark by formulating an analytical performance model for exploring application deployment scenarios in the edge continuum (Section V), and verify the model using our empirical results.

## II. Computing Models

*How should developers and infrastructure providers perform and support task offloading with cloud, edge, and endpoint resources?* Consulting computing models can help in answering this question as they offer a framework to explore the design space systematically. To execute tasks in the cloud for example, the cloud and serverless computing models can provide guidelines on how to leverage cloud resources to create application deployments [18]. For task offloading, many computing models exist, each having its assumptions on available infrastructure and application demands. Finding the right computing model for an application deployment thus requires a careful analysis of characteristics desired by the user and offered by computing models.

For this reason, in this section, we present a survey on task offloading computing models, building on previous work on computing models [19]–[22]. We synthesize key characteristics of state-of-the-art computing models and use the results of our survey to select five computing models key to navigating task offloading between endpoint and cloud resources.

[1]Established in 2011, the Cloud Group of the SPEC Research Group focuses on the general and specific performance issues associated with cloud operation, from traditional to new performance metrics, from workload characterization to modeling, from concepts to tools, from performance measurement processes to benchmarks. The work presented here is part of a larger, long-term activity within this group, focusing on understanding the systems principles of cloud and edge computing. The activity has started in 2019 and has resulted in several publications, which are available online and as open science artifacts. The group agrees with the publication of this article under the current co-authorship.

TABLE I: Comparison of key characteristics of 5 computing models for task offloading between cloud, edge, and endpoint. MC: Mist computing; EC: Edge computing; MEC: Multi-access edge computing; FC: Fog computing; MCC: Mobile cloud computing.

| Characteristic | MC | EC | MEC | FC | MCC |
|---|---|---|---|---|---|
| Offload target | Endpoint | Edge | Edge | Edge | Cloud |
| Compute capacity | Low | Moderate | Moderate | Moderate to high | High |
| Network latency | Low | Low | Low | Low to moderate | High |
| Network type | Wired, Wireless | Wired, Wireless | Wireless | Wired, Wireless | Wired |
| Architecture | Peer-to-peer | Distributed | Distributed | Hierarchical | Distributed |
| Offload service | Compute, storage | Compute | Compute | Compute, storage | Compute, storage |
| Operator | User | User, cloud provider | Network provider | Cloud provider | Cloud provider |
| Main use case | Peer-to-peer IoT processing | Real-time data processing | Mobile, network-aware data processing | Low latency cloud service provisioning | Compute- and storage-intensive tasks |

### A. Model Selection Process

In our survey, we have found 16 computing models that leverage a combination of cloud, edge, and endpoint devices and enable workload offloading between these devices. Endpoint devices live at the far end of the network, generate data, and are typically resource-constrained. Examples are sensor nodes, smart devices, and IoT devices (Figure 1). We have found significant overlap between computing models by analyzing their characteristics: Some computing models have been succeeded by others, some are a subset of more broadly oriented computing models, and some describe the same underlying behavior. We select five key models for further analysis; all are part of active research or used by industry and are the most prominent models in their part of the task offloading design space. The selected models are: mist computing [12], edge computing [23], multi-access edge computing [24], fog computing [25], and mobile cloud computing [16].

### B. Overview of Existing Computing Models

We provide a detailed explanation of each selected computing model using the edge continuum system model shown in Figure 1 and explain why these models have been selected over related computing models. A summary of our findings is presented in Table I, showing key differences between task offloading computing models using a variety of infrastructure and application characteristics.

*1) Mist computing:* Is concerned with forming a peer-to-peer network of endpoint devices, where resource-constrained and/or busy endpoints can offload workload to nearby endpoints with more processing power or storage capacity [12]. The computing models mobile crowdsourcing [26], peer-to-peer computing [27], and mobile crowd computing [28] describe similar behavior to mist computing. Mobile ad hoc cloud computing, also known as mobile edge-cloud computing, is limited to temporary, dynamic networks of endpoint devices [29]. Finally, with transparent computing, a cloud of endpoint devices is managed by edges and clouds [30].

*2) Edge Computing:* Enables the offloading of computation from endpoints to edge devices through low-latency networks. The edge can be connected to the cloud for further processing and storage [13], [31]. Related to edge computing is edge-centric computing, which focuses on applications with human interaction.

*3) Multi-access Edge Computing:* Previously known as mobile edge computing [32], MEC is concerned with augmenting cellular and wireless infrastructure with computing capacity to process workload from multiple endpoints connected via 4G or 5G [14], [24]. Contrary to edge computing, where edge devices can be owned by users or cloud providers, edge devices in multi-access edge computing are exclusively owned by network providers and standardized under the European Telecommunications Standards Institute (ETSI).

*4) Fog Computing:* Extends cloud services for task offloading to the edge to offer various cloud services to endpoints at low latency [2], [31]. The fog consists of many clusters of devices that can cooperate: From resource-constrained devices near users to micro data centers near the cloud, providing a trade-off between compute capacity and network latency. Offloading cloud services to micro data centers is the focus of cloudlet computing, a precursor of fog computing [8].

*5) Mobile Cloud Computing:* Endpoints offload compute tasks directly to the cloud [16]. The use of this computing model over edge/fog-related models is supported by the rapid increase in the number of cloud data centers constructed in the last decade, significantly decreasing the communication latency of most endpoints to their nearest cloud [33]. Mobile devices can be used to preprocess data to reduce network load, which is the focus of in situ computing [34]. Before the advent of modern clouds, mobile grid computing [35] supported compute offloading from endpoints to grids. Finally, osmotic computing combines mobile cloud computing and edge computing, advocating for a dynamic and seamless offloading process to cloud or edge [36].

## III. Edge Continuum Reference Architecture

Having reduced the number of key computing models for task offloading to five, we have significantly simplified the process of finding the right computing model for a particular application deployment. Finding the right model now requires comparing requirements from a particular application deployment to the features provided by the five models (Table I) and selecting the best fit. However, this approach of considering each model in isolation offers a restricted view of the task offloading design space. For example, data processing

applications may require computation services at the edge and storage services at the cloud [3]: Edge computing offers the former, and mobile cloud computing offers the latter, but no computing model explicitly offers these services combined. This shows that choosing an existing computing model with fixed characteristics for a particular application deployment has major limitations and creates confusion for users. In this section, we argue for a unified model that covers the entire design space of edge continuum deployments.

### A. Motivation for a Unified Reference Architecture

A computing model typically has a reference architecture that describes the components a computing model operates on and the interactions between these components. All five selected computing models have reference architectures, with the most prominent architectures listed in Section VI. Thus, to construct a unified computing model, we need to construct a unified reference architecture. The goal of building a unified reference architecture is to abstract away the specifications of the underlying hardware and peculiarities of specific computing models and compare more broadly the building blocks for a unified edge continuum. This is similar to cloud computing, which is not limited to specific use cases and unifies deployments in the cloud under a single model.

To design this architecture's components and their interactions, we need to analyze the commonalities and differences between the five computing models for task offloading. We can use the overlap between the models as the basis of the architecture, with the differences between the models becoming implementation details of the components [21]. For example, all selected computing models use resource managers to manage the complex stack of cloud, edge, and endpoint resources and the networks connecting them, but the specific implementation of these resource managers differs widely between application deployments. As such, resource managers should be a core component of the architecture, while the implementation of the resource manager should be an implementation detail of the component. While the implementation does affect application deployment characteristics such as end-to-end latency and resource utilization, it does not impact the core responsibilities and functioning of a resource manager or other components of a deployment.

The overlap between the five selecting computing models allows us to conceptualize a unified reference architecture for the edge continuum, which covers all concerns and mechanisms present in task offloading computing models and all resources in the continuum. By creating a unified architecture for the edge continuum, we present a single platform for research into cloud, edge, and endpoint resources, combining previous research efforts and allowing exploration in new research directions. We are the first to create such a unified architecture, extending previous work which looked at computing models in isolation (Section VI). While the term edge continuum has been used before [17], there has been no attempt to construct a reference architecture and show its utility.

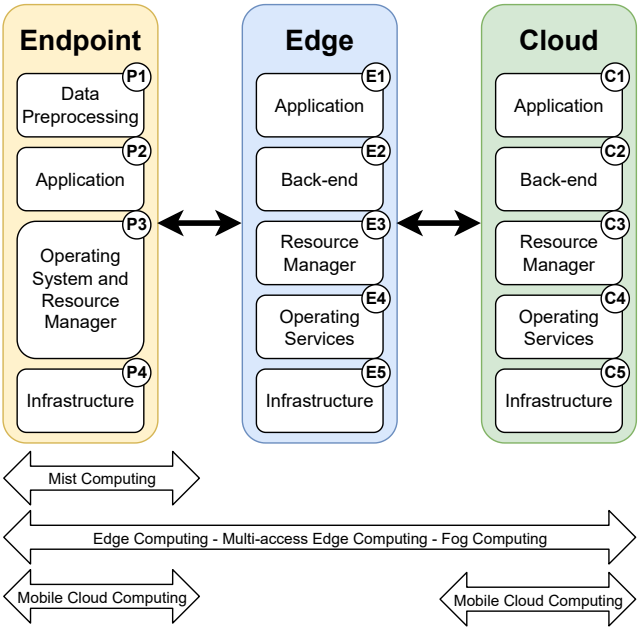


Fig. 2: Reference architecture for the edge continuum. The computing models are mapped to the parts of the architecture relevant to them.

### B. Architecture Overview

Having made a case for a unified edge continuum, we present our reference architecture for the edge continuum in Figure 2. The reference architecture consists of three tiers of systems: cloud, edge, and endpoint, with their associated components and responsibilities. The intuition behind this split is taken from a task offloading and data processing point of view: Endpoint devices create data streams and are the last connected components in the architecture, while cloud data centers are the top location where data can be pushed to for long-term storage and processing. Finally, the edge represents multiple hops of processing and storage in between endpoints and the cloud. The same split into three tiers holds up for other use cases such as content delivery networks, although responsibilities per tier will differ. The components of our architecture are well known and well studied: We borrow components from cloud computing as it provides general guidelines on data processing in clouds, which we can use to define data processing guidelines for the edge continuum. By using well-known components, we provide a unified architecture between different classes of devices and increase the architecture's usability. We provide examples of how to use our reference architecture in Section III-D.

### C. Architecture Components

In this section, we discuss the design of the architecture's components and the rationale behind their position in the reference architecture.

*1) Endpoint:* Endpoints are systems that live at the far end of the network. These are the last hop, or mile, in the architecture and typically generate and process data.

Examples are environment sensor nodes, smartphones and IoT devices. Key features of endpoints are: First, they are data sources, e.g., user input through online gaming [5], environment sensing [3], or video surveillance [6]. Second, they are resource-constrained, e.g., IoT devices such as sensors, embedded systems, and wearables [8]. Endpoint devices such as smartphones and self-driving vehicles contain more processing power and memory but are still constrained by their limited energy budget. Lastly, they may be mobile, e.g., smartphones, where location-sensitive services must migrate to the device's location. Compared to edge systems or cloud data centers, endpoints are typically single-tenant [37]; hence, OS-level resource management and multiplexing are often enough to meet the application needs. To support a unified reference architecture (that can support all computing models for the edge continuum), we have split up the endpoint responsibilities into the following four components:

- P1 **Data Preprocessing**: As endpoints are data sources that generate data, often there are capabilities to preprocess generated data according to a specific application domain. For example, in video surveillance, camera endpoints can support built-in object tracking or face detection [38], as can be the case for self-driving vehicles [7]. Such preprocessing capabilities can be built-in as a hardware accelerator or a software component available for use in high-level applications, thus freeing them to do low-level data preparations.
- P2 **Application**: Applications contain user-defined logic that runs on the endpoint to process the incoming data and make decisions. Typical logic in the application can be running heuristics, making offloading decisions, monitoring performance metrics, and/or triggering changes in the application deployments. User code can be supplemented with libraries such as TensorFlow Lite, which offer a programming model for specific application domains.
- P3 **Operating System and Resource Manager**: This component provides an interface between the endpoint's resources and the applications running on the device. Common examples of such components are Android, iOS, TinyOS, and QNX [39]. In many cases, these components are optimized for the deployed hardware and scenarios such as the presence or absence of certain hardware features, energy management, high priority for user interactions, etc.
- P4 **Infrastructure**: Infrastructure includes all physical compute, memory, network, and storage resources available to the operating system. The physical resources can be stationary or mobile. Unlike cloud data centers, the infrastructure at the endpoint can be owned and operated by application users, developers, or cloud providers. [2].

*2) Edge:* The reference architecture in Figure 2 shows that edge and cloud share the same high-level design. This is because both can run multi-tenant workloads on shared infrastructure and so need to offer similar services to users. However, uniquely at the edge, there should be support for application offloading both vertically (cloud to edge and back) and horizontally (from one edge system to another) [11]. The presence and absence of such capabilities determine what computing model one can use at the edge, e.g., edge devices can not communicate among themselves with edge computing, while they can with fog computing. These unique capabilities are implemented in the edge components. In order to encompass these different possibilities, we define the following components for edge systems:

- E1 **Application**: Edge applications are at the first step from the endpoints, and hence they are in the best position to make decisions regarding placements, offloading, and scheduling of application components to meet the application-specific objectives. For example, applications can choose to either do processing at the edge or preprocess data and offload more complex workloads to the cloud [10].
- E2 **Back-end**: Represents more general-purpose application execution frameworks such as TensorFlow Lite, MXNet, and WebAssembly runtime [40]. These back-end frameworks typically can manage application-specific memory, storage, communication, and workflows.
- E3 **Resource Manager**: Manages an edge system's application-independent *physical and virtual resources* (such as virtual machines and containers). These resource managers can be local or distributed, and their architecture determines what type of computing model one can run on the infrastructure.
- E4 **Operating Services**: Provides support to build distributed applications, and their responsibilities include (but are not limited to) communication [41], metadata management [42], consensus services [43], monitoring [44], storage services [45], etc.
- E5 **Infrastructure**: Similar to endpoints, compute, memory, and storage resources are provided to the resource manager. However, unlike endpoints, resources are split into physical and virtual resources. Direct access to hardware can be provided to users through bare-metal deployment, while virtualization technologies like virtual machines and containers abstract physical resources away to provide more flexibility and security for a slight performance penalty.

*3) Cloud:* The cloud consists of large-scale data centers managed by cloud service providers and is the core of the global networking infrastructure [18]. The cloud plays two roles in the edge continuum: The first is that of a central controller that manages multiple edge systems' resources and schedules workload on them [44], [46]. The controller has a global overview of the resources in the edge, as edge systems periodically update the cloud of their resource usage. Users upload edge applications to the cloud, which the cloud controller can schedule on specific edge systems. The global overview allows the central controller to schedule work more efficiently than a decentralized approach where each edge system manages itself. However, maintaining a global overview

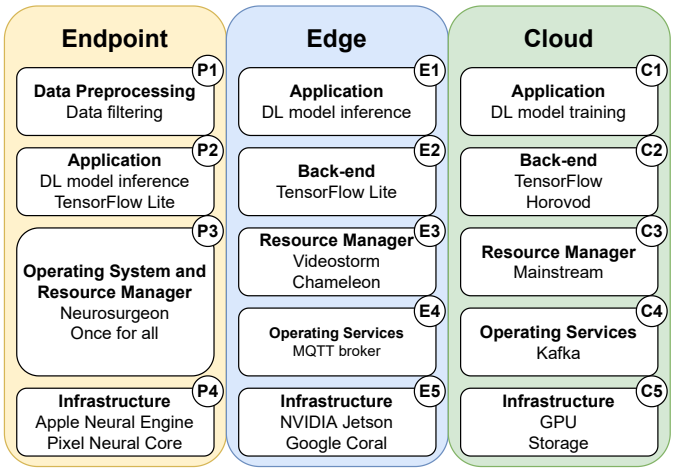


Fig. 3: Deep learning architecture.

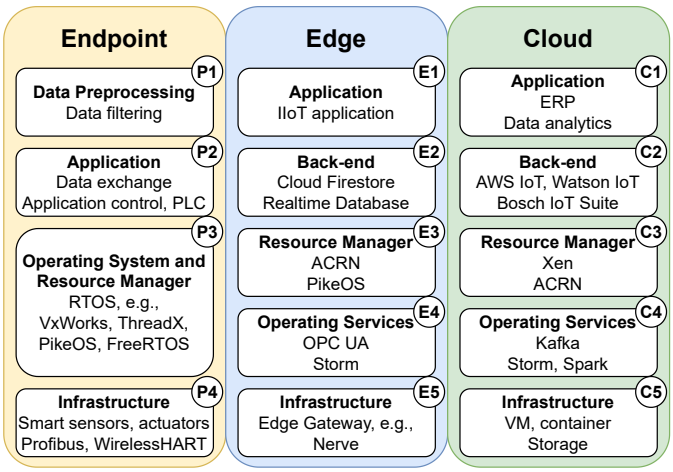


Fig. 4: Industrial IoT (IIoT) architecture.

and scheduling workload from the cloud adds extra overhead to the offloading process [9]. The second role is that of an offloading target that uses cloud computing to provide much more compute and storage capacity than edge [47]. Resources in the edge are limited [23], so cloud computing can be used to run resource-intensive applications like deep learning instead. However, applications that require low end-to-end latency can not leverage the cloud due to the increased communication latency compared to the edge [17].

The role of offloading target in the edge continuum is key to both edge and cloud; therefore, they share the same high-level design in our reference architecture. The components from edge, described in Section III-C2, can also be applied to cloud. A reference architecture for cloud computing has already been provided by Liu et al. [18].

### *D. Domain-specific Architectures*

In this section, we demonstrate how the unified reference architecture can be used to explore design trade-offs for application deployments in the edge continuum. Specifically, we create architectures for deep learning and industrial IoT (IIoT) use cases and provide example systems for each component. The architectures are visualized in Figures 3 and 4. We show that our architecture can guide users in how they can best use the edge continuum and infrastructure developers and cloud service providers in how they can design solutions for their clients. Moreover, we show that the uniform approach of our architecture allows for quick changes between deployments depending on changing application requirements.

*1) Deep Learning:* In deep learning, models are trained on large datasets to learn to accurately classify or regress a specific type of data, whereafter the trained model can be inferred to analyze new data. Deep learning is commonly used in data processing use cases such as recommendation systems [48] and voice and video analysis [49]. Model training and inference have different characteristics and can be executed on different devices in the edge continuum. Deciding where and how to deploy these tasks presents complex trade-offs and requires careful analysis. The unified reference architecture in Figure 3 can help with this, highlighting several possible systems for each architecture component, each providing its specific trade-offs. This way, developers and service providers can get an overview of what options are available for deep learning in the edge continuum, create custom deployments that suit their application's requirements, and switch between deployments if requirements change.

Deep learning model training requires a high compute and storage capacity and therefore is most often executed in clouds, offloading data for training from endpoints to the cloud using mobile cloud computing (component C1 in Figure 3). Depending on the application's requirements, such a deployment may be satisfactory: Services and platforms for machine learning like AWS SageMaker [50] (C2 and C3) can be used to create a performant machine learning pipeline. However, data required for model training may be privacy sensitive and thus can not always be offloaded to public clouds. Federated learning provides a solution as users train a local copy of a deep learning model on private data and share the deep learning model with other users (P2, P3, E2, E3). In this way, users profit from training done by other users while never sharing data, removing privacy issues.

While model inference is much less compute and storage intensive than model training, it still requires the use of specialized deep learning frameworks like Neurosurgeon [10] and models like MobileNet [51] to be deployable on edge or endpoint devices (P2, P3, E2, E3). These specialized frameworks and models present a complex trade-off between response time and model accuracy: By lowering the compute and storage requirements for model inference, model accuracy drops, lowering the application's performance, but the application can be deployed on user devices or edge devices close to the user, lowering response time to the user. For recommender systems, real-time user feedback may be required, so response time is preferred above model accuracy, while for video analytics the opposite may apply.

*2) Industrial IoT:* With the rise of Industry 4.0, endpoint devices like sensors and actuators have been increasingly integrated into industrial production processes to simplify automation, improve communication, and increase production without the need for human intervention [52]. Endpoint IIoT devices generate a large amount of data that needs to be processed, analyzed, and stored. These endpoints are connected to Programmable Logic Controllers (PLCs, component P2 in Figure 4); these are control systems for local control without support for advanced processing due to resource limitations. Thus, offloading to remote devices over a fieldbus or via wireless communication is required (P3). The choice of where to offload to depends then on several different aspects, including the geo-distribution of the data, the data rate, security and privacy issues, the required durability of the collected data, or simply the need for computation capabilities.

As many industry deployments include a vast amount of sensors and actuators, processing and storing the large amounts of generated data requires extensive resources. Therefore, public cloud offerings like AWS IoT and Bosch IoT suite (C2) can be a good fit for many deployments [52]. However, reliable real-time processing guarantees may be broken if the network infrastructure between the endpoints and cloud can not support the large data streams. Moreover, offloading sensitive data to third parties may introduce security and privacy concerns. On-premise cloud devices are therefore often used in IIoT deployments, delivering more stable performance by eliminating possible connectivity issues to remote public clouds and guaranteeing data privacy. Running a local cloud deployment is much more complex than offloading to a third party however, highlighting the importance of using a unified reference architecture in exploring deployment trade-offs.

In addition to clouds, edge gateways (E5) can be used to offer prompt response time and increased security. As data streams from endpoints and related compute tasks can be significant in IIoT deployments, cloud computing is fundamental in serving IIoT applications however. Therefore, platforms such as RACE have been developed that support cooperation between edge and cloud devices [53].

## IV. Deployment and Benchmarking Framework

In the design of our reference architecture, we showed that by changing the implementation of components in our architecture, we can create deployments for any task offloading computing model. To demonstrate this mechanic in practice, we synthesize an infrastructure deployment and benchmarking framework for the edge continuum in this section. This combination between infrastructure deployment and benchmarking, both highly configurable through a list of parameters, allows us to perform a systematic quantitative exploration of the edge continuum deployment space by performing performance analysis on architecture components. We first discuss the framework's design, then perform a selection of experiments to demonstrate its utility, and in Section V construct a performance model based on the benchmark's evaluation to enhance its performance analysis capabilities. The framework is open-sourced and available at https://github.com/atlarge-research/continuum.

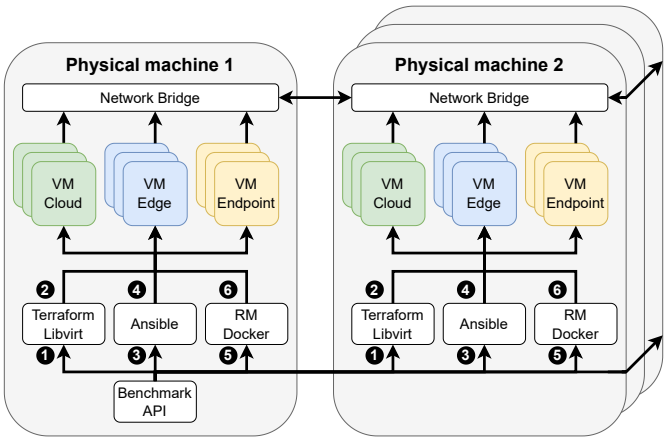


Fig. 5: Design of the deployment and benchmarking framework for the edge continuum.

### A. Framework Design and Interaction

Performing experiments in the edge continuum is challenging due to the wide range of networks and hardware in the continuum (Figure 1). Currently, no physical infrastructure is available that allows the exploration of all task offloading computing models and related deployments, only a selection [54]. As an alternative, devices and networks can be virtualized to emulate the edge continuum on commodity hardware. This emulation offers a flexible and easy-to-use method to explore the edge continuum's deployment space, as emulated hardware is configurable unlike physical hardware. Furthermore, such an emulated environment can be used to benchmark various deployment scenarios using components from our reference architecture. We have created such an emulated deployment and benchmarking framework and present its design in Figure 5. Our framework allows the emulation of many virtual machines across many physical machines, ranging from commodity hardware to specialized edge or endpoint hardware. This flexibility allows our framework to function as an emulated benchmarking environment on general-purpose hardware and as a benchmark on physical edge continuum resources. We offer a set of parameters to the user to interact with the framework and underlying systems on a high level. This interface lets users quickly switch between edge continuum deployments, helping them analyze design trade-offs. We present these parameters in Table II.

On the infrastructural level (components P4, E5, and C5 in Figure 2), users can specify the number of required cloud, edge, and endpoint devices that should be emulated, the specifications of the emulated devices, and the networks connecting the devices (component ❶ in Figure 5). For each device, the number of CPU cores and a quota per CPU can be defined, allowing a virtual machine to use a part of a CPU core, emulating resource-constrained devices (e.g., a quota

TABLE II: Selection of parameters offered by the framework.

| Parameter | Architecture Component | Description |
|---|---|---|
| Data generation frequency | P1 | Rate at which data is generated at endpoints. |
| Application | P1, P2, E1, E2, C1, C2 | Application to deploy and benchmark. |
| Resource manager | P3, E3, E4, C3, C4 | Resource manager to deploy in the continuum, together with related operating services. |
| Hypervisor | P4, E5, C5 | Virtual machine provider, e.g., QEMU. |
| Devices per tier | P4, E5, C5 | Number of cloud, edge, and endpoint devices to emulate. |
| Cores per device | P4, E5, C5 | Number of CPU cores to assign to each VM. |
| Quota per CPU | P4, E5, C5 | Emulated CPU cores can not consume more CPU time per fixed time interval than this quota. |
| Network per tier | P4, E5, C5 | Throughput and latency between emulated devices. |
| Machine addresses | P4, E5, C5 | IP addresses are required to enable emulation across multiple physical machines |

```
[infrastructure]
hypervisor = qemu
thread_pinning = True

# VM settings for cloud, edge, endpoint
devices_per_tier = 11,0,40
cores_per_device = 4,0,1
quota_per_cpu = 1.0,0,0.33

# Latency (ms): average,variability
cloud_to_cloud = 1,0
cloud_to_endpoint = 45,5

# Throughput (Mbit): average
cloud_to_cloud = 1000
cloud_to_endpoint = 7.5

machine_address = 192.168.1.1,192.168.1.2

[benchmark]
use_benchmark = True
data_generation_frequency = 5
application = image_classification
resource_manager = kubernetes
```

Listing 1: Framework configuration file for the cloud deployment presented in Figures 6 and 7.

of 0.5 defines the use of half a CPU core). Currently, we support QEMU/KVM [55] as the primary hypervisor, generate configuration files for it based on user input, and launch emulated hardware across one or multiple physical machines (❷). Then, using Linux Traffic Control, we emulate network latency and throughput between virtual machines. We do not standardize the infrastructure or any component of the reference architecture in our framework: Any other hypervisor, operating service, resource manager, or application can be used with our framework besides the ones currently supported.

As the next step, operating services and resource managers are installed in the provisioned virtual machines (P3, E3, E4, C3, C4) through the software deployment tool Ansible [56] (❸ and ❹). Currently, Kubernetes is supported as cloud resource manager and KubeEdge as edge resource manager. We plan to extend this offering to resource managers based on serverless technology in the future. Finally, on the application level (P1, E1, C1), we support data processing applications consisting of a data generation component located on endpoints and a data processing component on cloud, edge, or endpoint devices. This allows for great flexibility in deployments and supports all our surveyed computing models. Finally, applications on cloud and edge get deployed through their respective resource manager, while endpoint applications are deployed directly as a container via Docker, aligning with our reference architecture (❺ and ❻). Application instances can communicate through a resource manager or operating services such as MQTT.

We provide an example configuration of our framework in Listing 1, displaying a configuration for a large-scale mobile cloud computing deployment which we will later use in our evaluation in Figures 6 and 7. This configuration shows all parameters the framework has to offer its users: Those that interact with our reference architecture (Table II) and those that enable extra features such as thread pinning of emulated CPU cores for performance isolation. Note that the use of the benchmark is optional in our framework, as indicated by the $use\_benchmark$ parameter: Users can decide to only provision edge continuum infrastructure to explore system and application deployments by hand.

### B. Evaluation Overview

We implement the proposed framework design and conduct several experiments to explore a selection of deployment scenarios using cloud, edge, and endpoint devices. More specifically, we attempt to answer three fundamental questions:

1) *First, does our framework allow exploring different complex deployments in the edge continuum?* We create complex deployments for task offloading to cloud, edge, and endpoint devices (Table III), show how users can quickly switch between configurations (Listing 1), and prove the functionality of our framework (Figure 6).
2) *Secondly, how can our framework help explore design trade-offs for real-time applications?* For various task offloading scenarios, we provide a breakdown of end-to-end latency, consisting of communication latency between data source and offloading target and data processing time on the offloading target (Figure 7).
3) *Lastly, how can our framework guide offloading decisions under various workload levels?* We explore system load when increasing the number of data-generating devices connected per offloading target, demonstrating how appli-

TABLE III: Framework parameters for the deployments used in our evaluation.

| Parameter | Cloud | Edge-Large | Edge-Small | Mist |
|---|---|---|---|---|
| Resource manager | Kubernetes | KubeEdge | KubeEdge | - |
| Worker location | Cloud | Edge | Edge | Endpoint |
| Workers | 10 | 10 | 10 | 10 |
| Worker cores | 4 | 4 | 2 | 2 |
| Worker quota | 1.0 | 1.0 | 0.75 | 0.5 |
| Endpoints per worker | 4 | 4 | 2 | 1 |
| Network latency (ms) | 45 | 30 | 7.5 | 7.5 |
| (#cloud, #edge, #endpoint) | (11, 0, 40) | (1, 10, 40) | (1, 10, 20) | (0, 0, 20) |

cation performance changes when offloading targets are overloaded (Figure 8).

### C. Experimental Setup

For our experiments, we implement a machine learning application consisting of a data generation component deployed on endpoints and a data processing component deployed on a cloud, edge, or endpoint offloading target. For the data generation, we emulate a camera that generates a configurable number of images per second, and for the data processing, we emulate a processing unit performing object detection on these images. After data is processed, the resulting output is sent back to the original data source device. MQTT, a lightweight pub/sub messaging protocol, handles communication between the two application components. This generalizable setup should result in the machine learning application behaving similarly to data processing applications from different domains. We fix the data generation rate for our experiments to five images per second and set the bandwidth between data-generating endpoints and offloading targets to 8 Mbit/s, an average throughput value for 4g networks [57].

The experiments are performed on three Intel Xeon Silver 4210R machines using QEMU 6.1.0, Libvirt 6.0.0, Kubernetes 1.21.0, and KubeEdge 1.8.1. Figures 6 and 8 report the system load of the devices running data processing tasks. The system load is calculated by comparing the number of images processed per second on the device to the number of images being offloaded to the device. With this definition, a system load of over 100 percent indicates that the processing demand created by data generating endpoints exceeds the processing devices' processing speed. Figures 7 and 8 report end-to-end latency, the time between a data element being generated and the processed output being available to the original data source device. It includes data processing time and communication latency between data source and offloading target. We let the data generation component run for 5 minutes per experiment, repeat each experiment three times, and report the average and standard deviation values between runs. We pin each emulated CPU core to a physical CPU core, guaranteeing that no physical machine is oversubscribed with running virtual machines, resulting in improved performance isolation and more stable results.

### D. Exploration of Application Deployments

For our first experiment, we demonstrate the infrastructure deployment framework's ability to create large-scale edge continuum deployments and the functionality of the benchmark by measuring system load when offloading workload generated on endpoint devices to cloud, edge, and endpoint workers. Table III lists the configurations of the deployments we use in this experiment. The general trend is that the closer the offloading target is to the data source in terms of communication latency, the lower the worker's compute capacity and the fewer clients it can serve without overloading.

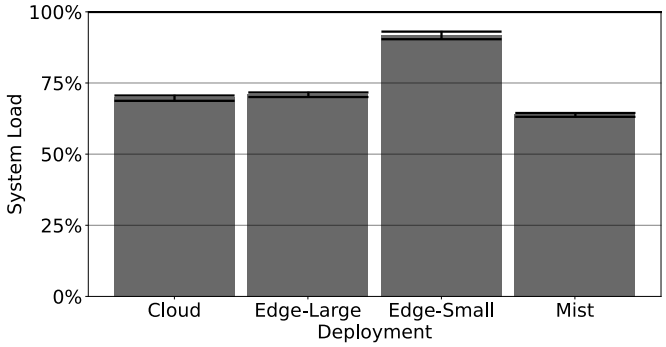


Fig. 6: Large scale deployment for edge/cloud/endpoint.

In Listing 1, we provide a user-defined configuration file showing the exact setup of the cloud deployment mentioned in Table III. From this configuration file, the following parameters need to be updated to create the Edge-Large deployment where a micro data center at the edge is emulated: The number of devices per tier and their settings, the network latency and throughput, and the resource manager (KubeEdge is the preferred resource manager for edge deployments). In this way, by changing a few lines of code, entirely new deployments can be created by users, aiding deployment exploration.

We deploy our machine learning application on the four deployments and measure the average system load on the offloading targets during application execution. The results in Figure 6 show that all generated data can be processed in real-time as the system load never exceeds 100 percent. Real-time processing is achieved by adapting the workload in terms of number of data-generating endpoints connected per worker to the compute resources of the offloading target. We show in Section IV-F how users can discover deployments that enable real-time processing. The variance in system load between runs is minimal due to our framework's thread pinning feature.

### E. End-to-end Latency Analysis

To further enhance the performance analysis capabilities of our benchmark, we measure end-to-end latency for our four deployments. By analyzing end-to-end latency, users can discover if offloading targets are overloaded and if any latency requirements are met. We break end-to-end latency down into two components: Communication latency between data source and offloading target and processing time on the offloading target. We add queuing delays in the next section.

Our results in Figure 7 show a trend of a decreased communication latency when the offloading target moves closer

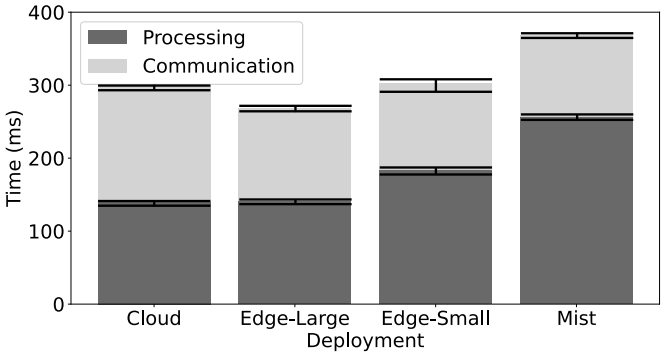


Fig. 7: Breakdown of the end-to-end latency per deployment.

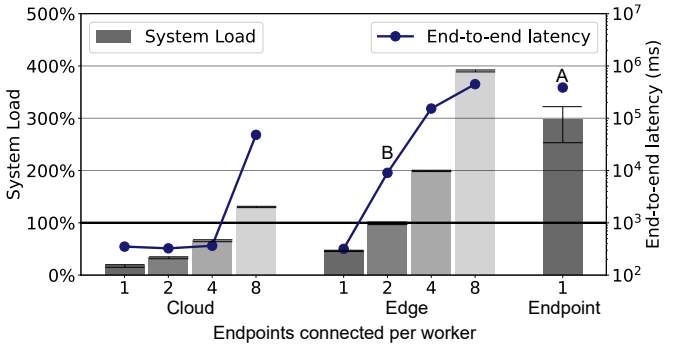


Fig. 8: System load when processing data, with an increasing number of endpoints connected to one processing device. Experiments A and B are used for the performance model examples in Equations 3 and 4 respectively.

to the data source and an increased processing time due to the closer offloading targets being more resource-constrained. These results match the deployment configurations as defined in Table III. An interesting find is that for this particular application deployment, processing time increases much faster than network latency decreases when moving the offloading target closer to the data source, resulting in a higher end-to-end latency for the mist computing deployment compared to the cloud deployment, something that might seem counter-intuitive at first. This behavior depends on the size of the offloaded data and the computational intensity of the processing application; other data processing applications might show different trends. The Edge-Large deployment achieves the lowest end-to-end latency, which uses similar compute resources to the cloud deployment for fast processing latencies, emulating a micro data center, but with lower communication latency as it is positioned at the edge. The deviation in end-to-end latency between experiment repetitions is again minimal.

### *F. System Load Analysis*

For the final experiment, we deploy a single worker at the cloud or edge and connect them to one to eight data-generating endpoints. By connecting more endpoints per worker, the system load of the worker increases, as visualized in Figure 8. We compare the offloading performances to a baseline of an endpoint processing its own generated data. The cloud workers use four cores with a quota of 1.0, the edge workers use two cores with a quota of 0.66, and the endpoint devices use one core with a quota of 0.33.

Given these settings, the results are as expected: System load is lowest when offloading to cloud due to it having plenty of compute capacity, and offloading in general performs better than local processing on resource-constrained endpoints. The interesting part is that we can quantify under what workload we can still perform real-time processing, which equals a system load of 100 percent or less in Figure 8. With the current configuration, endpoints can not perform local processing in real-time, for example. We also include end-to-end latency in this experiment and see that latency explodes when system load exceeds 100 percent. This is because offloaded data starts queuing up at the offloading target when the system load exceeds 100 percent, dominating processing and communication overhead in the end-to-end latency. It tells users that either extra compute resources need to be added or generated workload needs to be decreased; otherwise, only a portion of generated data can be processed on time. Variance is minimal for all experiments but the endpoint baseline as it suffers from performance variability caused by the low CPU quota of 0.33, causing the VM to be context switched often.

## V. Analytical Performance Model

For our final contribution, we enhance the performance analysis capabilities of our edge continuum benchmark by introducing a simple to use, first-order analytical performance model that predicts if applications can be offloaded to cloud or edge, should be processed locally on endpoints, or are not suited for deployment in the edge continuum. The performance model provides a clear overview of how user-defined deployment scenarios perform in the edge continuum, what deployment models support the user's deployment scenario, and what can be done to improve performance. Such an overview is presented in Figure 9. The performance model is part of the benchmark suite and can be executed using provided scripts.

### *A. Model Description*

To predict if applications can be executed in the edge continuum, we need to check if the available network and compute resources in the continuum satisfy an application's data and processing requirements. Equations 1 and 2 describe our performance models for local execution on endpoints and offloading to edge or cloud. Only when the available systems meet all data and processing requirements will the performance models deem viable execution in the edge continuum.

We assume long-running data processing applications are used, such as the machine learning application used in the benchmark evaluation, so we ignore startup and clean-up overheads when offloading tasks. This scenario is common in edge data processing as sensors and other data-generating devices like cameras are in constant use.

$$Local = \begin{cases} 0 & \text{if } \frac{T_{proc}}{C_e \times Q_e} > P \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

$$Offload = \begin{cases} 0 & \text{if } \frac{T_{proc} \times E}{C_o \times Q_o} > P \\ 0 & \text{if } \frac{T_{pre}}{C_e \times Q_e} > P \\ 0 & \text{if } D > B \\ 1 & \text{otherwise} \end{cases} \quad (2)$$

where:

T_proc = Processing time per data element (sec)
T_pre = Preprocessing time per data element (sec)
C = CPU cores of endpoint $e$ or offloading target $o$
Q = CPU quota of endpoint $e$ or offloading target $o$
P = Data generation interval (sec)
E = Endpoints connected to offloading target
D = Data generated per second per endpoint (Mbps)
B = Bandwidth to offloading target (Mbps)

Given this context, we can describe our performance models in detail. The only requirement for local execution on endpoints (Equation 1) is that endpoints must process a data element before the next data element is generated ($P$). To model this, we benchmark the processing time per element $T_{proc}$ using a single CPU core and correct it with the number of cores $C$ and CPU quota $Q$ used by the endpoint device. An application is unsuitable for deployment on endpoints if it can not satisfy the processing requirement. The processing workload should be decreased, or more powerful hardware should be used to make deployment possible again.

Three conditions must be met for offloading to edge or cloud (Equation 2). First, edge or cloud devices should process each data element in time similarly to the endpoint performance model. However, with offloading, multiple endpoints $E$ can be connected to a single edge or cloud device, increasing the workload. Second, endpoints should preprocess each data element before the next element is generated. We model this by benchmarking the preprocessing time per data element $T_{pre}$ using a single core and correct this with the number of cores and CPU quota used by the endpoint device. Finally, the network between endpoint and offloading target should handle generated data stream. Given the size of a single data element and the number of data elements generated per second, the total amount of data generated per second per endpoint $D$ can be calculated and compared to the available bandwidth to the offloading target $B$.

### B. Example Calculation

To verify our performance models, we use a selection of the benchmark results, marked as *A* and *B* in Figure 8. We test if each condition of the performance model is satisfied and compare the system load from the benchmark to the prediction of the performance model. For both examples, the data generation interval $P$ is set to 0.2 seconds as endpoints generate five images per second in all benchmark experiments.

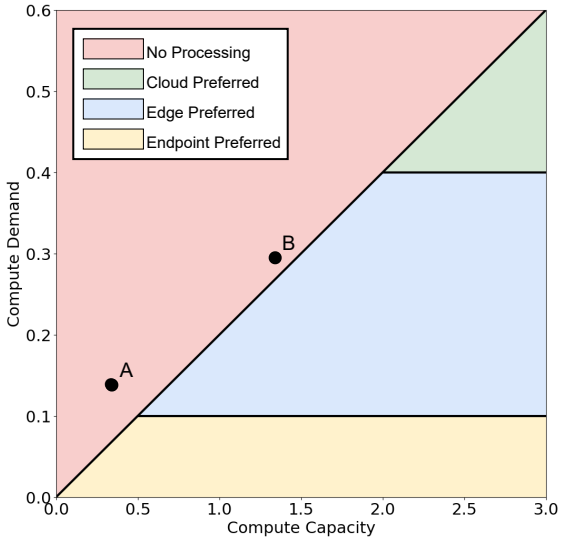


Fig. 9: Exploring preferred deployment models for the performance model examples from Equations 3 (A) and 4 (B).

$$Local = \begin{matrix} \frac{0.14}{1 \cdot 0.333} & < 0.2 \\ 0.42 & > 0.2 \end{matrix} \quad (3)$$

For the first example we use the local deployment performance model in Equation 3 and a corresponding benchmark deployment marked as *A* in Figure 8. After considering the endpoint's compute capacity, the processing time per data element is 0.42 seconds, making this deployment nonviable. With this, the system load of just over 200 percent is close to the benchmark result.

$$Offload = \begin{matrix} \frac{0.15 \times 2}{2 \times 0.66} & > 0.2 & \text{and} & \frac{0.001}{1 \times 0.33} & > 0.2 \\ 0.23 & > 0.2 & \text{and} & 0.0003 & < 0.2 \end{matrix} \quad (4)$$

For the second example, we use the offloading performance model in Equation 4 and the edge configuration using two cores and two connected endpoints marked as *B* in Figure 8. We consider the preprocessing and processing conditions, omitting the network condition as we set the network up not to drop data for the experiments. With little to no preprocessing happening on endpoints, its impact on performance is minimal, as expected. The processing part of the performance model, however, predicts that offloading processing tasks to edge is not possible, with an expected system utilization of just over 100 percent, while the benchmark in Figure 8 shows that offloading is possible with a system load of 97 percent. As both the performance model and the benchmark predict a system load close to 100 percent, this result should be interpreted instead as real-time workload offloading to edge being barely possible or barely not possible, with the advice to either decrease workload or increase resources.

TABLE IV: Selection of reference architectures mapped to computing models for task offloading. MC: Mist computing; EC: Edge computing; MEC: Multi-access edge computing; FC: Fog computing; MCC: Mobile cloud computing. Symbols: ●: Present; ○: Not present.

| Authors | MC | MEC | EC | FC | MCC |
|---|---|---|---|---|---|
| Yogi et al. [58] | ● | ○ | ○ | ○ | ○ |
| ETSI [24] | ○ | ● | ○ | ○ | ○ |
| ECC [23] | ○ | ○ | ● | ○ | ○ |
| IIC [4] | ○ | ○ | ● | ○ | ○ |
| Intel, SAP [59] | ○ | ○ | ● | ○ | ○ |
| OpenNebula [60] | ○ | ○ | ● | ○ | ○ |
| Sittón-Candanedo et al. [61] | ○ | ○ | ● | ○ | ○ |
| Qinglin et al. [62] | ○ | ○ | ● | ● | ○ |
| Willner et al. [63] | ○ | ○ | ● | ● | ○ |
| Mahmud et al. [64] | ○ | ○ | ○ | ● | ○ |
| OpenFog Consortium [25] | ○ | ○ | ○ | ● | ○ |
| Pop et al. [65] | ○ | ○ | ○ | ● | ○ |
| Dinh et al. [16] | ○ | ○ | ○ | ○ | ● |
| **Edge continuum (this work)** | ● | ● | ● | ● | ● |

### C. Offloading Benefits

To conclude the performance analysis, we present a recommendation on application execution in the edge continuum, similar to a Roofline model, in Figure 9. Here point A represents the local processing example from Equation 3, and B the offloading example from Equation 4. An application can only be executed in the edge continuum if the compute capacity of the processing device $C \times Q$ exceeds the processing requirement of the application $T_{proc}$. Using our performance model, the exact difference between the two is determined by the data generation interval $P$, which also determines the slope of the dividing line between no processing and processing in Figure 9. If processing in the edge continuum is possible, we prefer deployment as close to the user as possible; this is often more desirable in terms of end-to-end latency, privacy, energy efficiency, etc. The division between endpoint, edge, and cloud is based on compute capacity, where we set the maximum compute capacity of endpoints to 0.5 and of edge devices to 2.0. The user can change this depending on the specifications of its endpoint and edge devices.

For these two particular examples, we see in Figure 9 that deployment for the local processing example is impossible, while it is possible for the offloading example, preferably to the edge. This method allows us to explore the edge continuum deployment space and determine the best deployment for specific applications.

## VI. Related Work

In this section, we present previous work related to our contributions. We introduce the first reference architecture to consider the entire edge continuum in Section III, encompassing previous work which considers cloud, edge, and endpoint computing models in isolation (Table IV). To the best of our knowledge, only previous work from Qinglin et al. and Willner et al. has discussed the possibility of combining different computing models, however, this is limited to only edge and fog computing.

TABLE V: Comparison of characteristics of selected emulation and benchmarking frameworks for cloud and edge.

| Characteristic | Infrastructure [66] | Infrastructure [67] | Benchmark [69] | Benchmark [70] | Benchmark [71] | This work |
|---|---|---|---|---|---|---|
| Considers cloud resources | ● | ● | ● | ● | ● | ● |
| Considers edge resources | ● | ● | ● | ● | ● | ● |
| Considers endpoint resources | ● | ● | ● | ● | ● | ● |
| Considers network resources | ● | ● | ● | ● | ● | ● |
| Configurable compute resources | ● | ● | ○ | ○ | ○ | ● |
| Configurable networks | ● | ● | ○ | ○ | ○ | ● |
| Configurable resource managers | ○ | ○ | ○ | ○ | ○ | ● |
| Configurable operating services | ○ | ○ | ○ | ○ | ○ | ● |
| Application benchmarking | ● | ● | ● | ● | ● | ● |
| Supports all computing models | ○ | ○ | ○ | ○ | ○ | ● |

We design and implement a deployment and benchmarking framework for the edge continuum to explore the continuum's deployment space. On infrastructure provisioning and emulation, closest to our work, Symeonides et al. [66] present Fogify, a framework for emulating fog resources, and Hasenburg et al. present a similar framework with MockFog [67]. These systems are part of a larger class of cloud, edge, and endpoint resource emulation and simulation frameworks [68]. All support limited computing models and therefore are restricted in resources and deployments that can be emulated, unlike our framework, which enables emulation of all resources spawning the entire edge continuum. Additionally, by emulating virtual machines instead of isolation methods such as containers, our framework allows the configuration and benchmarking of architecture components other than applications. For example, users can currently switch between the Kubernetes and KubeEdge resource managers or add a new resource manager; the same configurability applies to operating services. We show the limitations of related work by comparing it to our framework in Table V.

On benchmarking edge continuum resources and systems, closest to our work, Kimovski et al. [69] propose a benchmarking framework spawning edge, cloud, and fog ; we add to it DeFog [70] and DeathStarBench [71]. These benchmark tools present a larger class of benchmarking tools for cloud, edge, and endpoint resources [72]. We improve upon these systems by offering more resources and deployment models to benchmark and allowing greater customization of compute and network resources. We argue that the coupling of infrastructure emulation and benchmarking that our framework offers is key in efficiently exploring the design space of edge continuum deployments and is not present in any related benchmark frameworks. We present a detailed comparison in Table V.

On modeling the performance of edge continuum systems, closest to our work, Majeed et al. [73] do performance modeling for workload offloading in the fog; we add to it work on network modeling from Ali-Eldin et al. [74]. These works provide detailed performance models for specific edge continuum deployments, unlike our first-order model, which can be applied to many models. Furthermore, our performance model is accompanied by an edge continuum benchmark which can be used to iterate on application deployment configurations quickly.

## VII. Conclusion and Ongoing Work

In this paper, we provide a survey on 16 different computing models that define how to offload tasks between cloud, edge, and endpoint resources. Our analysis reveals that even though these computing models are often presented in isolation, there is significant overlap. Given this insight, we argue for a unified model that considers all resources in the edge continuum. With this unified model, application and system development is no longer limited to the assumptions posed by a single computing model, opening up future advancements in research and engineering. We synthesize a reference architecture for the edge continuum and identify key building blocks of cloud, edge, and endpoint deployments. We highlight the architecture's utility for developers and service providers by creating domain-specific architectures for deep learning and industrial IoT and demonstrate how it exposes design trade-offs for each architecture component. We then perform a systematic quantitative exploration of the edge continuum deployment space by designing a deployment and benchmarking framework for edge continuum applications and investigating the performance of individual components of the architecture. We also introduce an analytical first-order performance model to enhance the performance analysis capabilities of the benchmark and use it to analyze multiple application deployment scenarios such as local processing on endpoints and offloading to cloud or edge. Using this model and framework, developers and infrastructure providers can explore the design space of the edge continuum to optimize deployments specific to their use case. The work on this reference architecture has been conducted in the SPEC-RG Cloud group. The deployment and benchmarking framework is open-source, under active development, and available at https://github.com/atlarge-research/continuum.

## References


[1] Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, 2010.

[2] AWS, "AWS Greengrass," https://aws.amazon.com/greengrass/, 2021, accessed: 2021-05-12.

[3] Vasisht *et al.*, "Farmbeats: An IoT platform for data-driven agriculture," in *NSDI*, 2017.

[4] Tseng *et al.*, "Introduction to edge computing in IIoT," Tech. Rep., 2018.

[5] Zhang *et al.*, "Improving cloud gaming experience through mobile edge computing," *IEEE Wirel. Commun.*, vol. 26, no. 4, 2019.

[6] Khan *et al.*, "Deep unified model for face recognition based on convolution neural network and edge computing," *IEEE Access*, vol. 7, 2019.

[7] Lin *et al.*, "The architectural implications of autonomous driving: Constraints and acceleration," in *ASPLOS*, 2018.

[8] Satyanarayanan *et al.*, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, 2009.

[9] Xiong *et al.*, "Extend cloud to edge with KubeEdge," in *IEEE/ACM Symposium on Edge Computing*, 2018.

[10] Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ASPLOS*, 2017.

[11] Mortazavi *et al.*, "Cloudpath: a multi-tier cloud computing framework," in *IEEE/ACM Symposium on Edge Computing*, 2017.

[12] Preden *et al.*, "The benefits of self-awareness and attention in fog and mist computing," *Computer*, vol. 48, no. 7, 2015.

[13] Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, 2017.

[14] Taleb *et al.*, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 3, 2017.

[15] Bonomi *et al.*, "Fog computing and its role in the internet of things," in *MCC*, 2012.

[16] Hoang *et al.*, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wirel. Commun. Mob. Comput.*, vol. 13, no. 18, 2013.

[17] Linux Foundation, "State of the edge 2021," https://project.linuxfoundation.org/hubfs/LF2021, accessed: 2021-06-06.

[18] Liu *et al.*, "NIST cloud computing reference architecture," *NIST special publication*, vol. 500, no. 2011, 2011.

[19] Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, 2019.

[20] Li *et al.*, "Edge-oriented computing paradigms: A survey on architecture design and system management," *ACM Comput. Surv.*, vol. 51, no. 2, 2018.

[21] Mouradian *et al.*, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 1, 2018.

[22] Taleb *et al.*, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 3, 2017.

[23] Edge Computing Consortium (ECC) and Alliance of Industrial Internet (AII), "Edge computing reference architecture 2.0," Tech. Rep., 2017.

[24] ETSI, "Multi-access edge computing (MEC); framework and reference architecture," *ETSI, DGS MEC*, vol. 3, 2019.

[25] OpenFog Consortium, "OpenFog reference architecture for fog computing," Tech. Rep. OPFRA001, 2017.

[26] Ren *et al.*, "Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions," *IEEE Commun. Mag.*, vol. 53, no. 3, 2015.

[27] Milojicic *et al.*, "Peer-to-peer computing," Tech. Rep., 2002.

[28] Fernando *et al.*, "Honeybee: A programming framework for mobile crowd computing," in *MobiQuitous*, vol. 120, 2012.

[29] Drolia *et al.*, "The case for mobile edge-clouds," in *UIC/ATC*, 2013.

[30] Ren *et al.*, "Serving at the edge: A scalable IoT architecture based on transparent computing," *IEEE Netw.*, vol. 31, no. 5, 2017.

[31] Shi *et al.*, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, 2016.

[32] ETSI, "Mobile edge computing (MEC); framework and reference architecture," *ETSI, DGS MEC*, vol. 3, 2016.

[33] Corneo *et al.*, "Surrounded by the clouds: A comprehensive cloud reachability study," in *WWW*, 2021.

[34] Li *et al.*, "Towards sustainable in-situ server systems in the big data era," in *ISCA*, 2015.

[35] Guan *et al.*, "A grid service infrastructure for mobile devices," in *SKG*, 2005.

[36] Villari *et al.*, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Comput.*, vol. 3, no. 6, 2016.

[37] Satyanarayanan *et al.*, "The computing landscape of the 21st century," in *HotMobile*, 2019.

[38] Hu *et al.*, "The case for offload shaping," in *HotMobile*, 2015.

[39] Hill *et al.*, "System architecture directions for networked sensors," in *ASPLOS*, 2000.

[40] Gadepalli *et al.*, "Sledge: a serverless-first, light-weight wasm runtime for the edge," in *Middleware*, 2020.

[41] Light, "Mosquitto: server and client implementation of the MQTT protocol," *J. Open Source Softw.*, vol. 2, no. 13, 2017.

[42] Erwin, "Erwin edge," https://www.erwin.com/products/, 2021, accessed: 2021-05-30.

[43] Hao *et al.*, "Edgecons: Achieving efficient consensus in edge computing networks," in *HotEdge*, 2018.

[44] Tencent, Intel, VMware, Huya, Cambricon, Captialonline, and Meituan, "Superedge," https://github.com/superedge/superedge, 2021, accessed: 2021-06-06.

[45] Gupta and Ramachandran, "Fogstore: A geo-distributed key-value store guaranteeing low latency for strongly consistent access," in *DEBS*, 2018.

[46] Wang *et al.*, "ENORM: A framework for edge node resource management," *IEEE Trans. Serv. Comput.*, vol. 13, no. 6, 2020.

[47] Microsoft, "Microsoft Azure IoT," https://azure.microsoft.com/en-us/services/iot-hub/, 2021, accessed: 2021-05-18.

[48] Wang and Wang, "Improving content-based and hybrid music recommendation using deep learning," in *MM*, 2014.

[49] George *et al.*, "Towards drone-sourced live video analytics for the construction industry," in *HotMobile*, 2019.
[50] AWS, "AWS SageMaker," https://aws.amazon.com/sagemaker/, 2021, accessed: 2021-05-17.
[51] Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018.
[52] Al-Gumaei *et al.*, "A survey of internet of things and big data integrated solutions for industrie 4.0," in *ETFA*, 2018.
[53] Chandramouli *et al.*, "RACE: real-time applications over cloud-edge," in *SIGMOD*, 2012.
[54] Senel *et al.*, "Edgenet: A multi-tenant and multi-provider edge cloud," in *EdgeSys*, 2021.
[55] Bellard, "QEMU, a fast and portable dynamic translator," in *ATC*, 2005.
[56] Hochstein and Moser, *Ansible*. O'Reilly Media, 2017.
[57] Raca *et al.*, "Beyond throughput: a 4g LTE dataset with channel and context metrics," in *MMSys*, 2018.
[58] Yogi *et al.*, "Mist computing: Principles, trends and future direction," *CoRR*, vol. abs/1709.06927, 2017.
[59] Intel and SAP, "IoT joint reference architecture from intel and sap," Tech. Rep., 2018.
[60] OpenNebula, "Edge cloud architecture - white paper," Tech. Rep. 2.0, 2021.
[61] Sittón-Candanedo *et al.*, "A review of edge computing reference architectures and a new global edge proposal," *FGCS*, vol. 99, 2019.
[62] Qi and Tao, "A smart manufacturing service system based on edge computing, fog computing, and cloud computing," *IEEE Access*, vol. 7, 2019.
[63] Willner and Gowtham, "Toward a reference architecture model for industrial edge computing," *IEEE Commun. Stand. Mag.*, vol. 4, no. 4, 2020.
[64] Mahmud *et al.*, "Cloud-fog interoperability in IoT-enabled healthcare solutions," in *ICDCN*, 2018.
[65] Pop *et al.*, "The FORA fog computing platform for industrial IoT," *Inf. Syst.*, vol. 98, 2021.
[66] Symeonides *et al.*, "Fogify: A fog computing emulation framework," in *IEEE/ACM Symposium on Edge Computing*, 2020.
[67] Hasenburg *et al.*, "Mockfog 2.0: Automated execution of fog application experiments in the cloud," *CoRR*, 2020.
[68] J. Taheri and S. Deng, *Edge Computing: Models, Technologies and Applications*. The Institution of Engineering and Technology (IET), 2020.
[69] Kimovski *et al.*, "Cloud, fog, or edge: Where to compute?" *IEEE Internet Comput.*, vol. 25, no. 4, 2021.
[70] McChesney *et al.*, "Defog: fog computing benchmarks," in *IEEE/ACM Symposium on Edge Computing*, 2019.
[71] Gan *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *ASPLOS*, 2019.
[72] Varghese *et al.*, "A survey on edge performance benchmarking," *ACM Comput. Surv.*, vol. 54, no. 3, 2021.
[73] Majeed *et al.*, "Modelling fog offloading performance," in *ICFEC*, 2020.
[74] Ali-Eldin *et al.*, "The hidden cost of the edge: a performance comparison of edge and cloud latencies," in *SC*, 2021.