Serverless Computing at the Edge in Precise Agriculture

Edgardo J. Reinoso Campos
Vrije Universiteit Amsterdam
The Netherlands
e.j.reinosocampos@student.vu.nl

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

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

Abstract

Serverless and edge computing are two technological innovations which have improved efficiency in computation across many fields, in particular agriculture. Serverless computing allows developers to write and run code without the need for servers or infrastructure, while edge computing brings processing power closer to where data is generated, reducing latency and improving performance. Precise Agriculture is an example of how these two technologies play a key role in improving efficiency and productivity. By leveraging serverless computing, applications can easily be developed and deployed for automating tasks such as crop monitoring, irrigation management, and livestock tracking. As a result, this enables farmers to make real-time decisions and optimize their operations based on data insights. Edge computing, on the other hand, allows data processing at the edge of the network. This reduces the time to process data and enables faster decision-making, which is critical in agriculture. For example, edge computing can be used to analyze sensor data from soil moisture sensors, weather stations, and drones, allowing farmers to make decisions about when to plant, water, and harvest their crops. However, in order for both of these technologies to work in such conditions like open fields with restraint connectivity, there are some evaluations that should be considered in the serverless structure. Therefore, this survey will explore the different optimizations that can be applied with respect to the serverless execution environment and resource management in order to make computation more efficient at the edge.

Keywords

Serverless Computing, Edge Computing, Precise Agriculture

1 Introduction

Edge computing has been developed to extend the processing capabilities from the cloud closer to where data is originated. This distributed computing model leverage smaller devices for performing computation locally without the need of relying solely on centralized data centers or cloud computing. In this model, data processing and analysis are performed on the edge of the network, near the source of data, rather than sending it over the network to a remote data center for initial processing.

This model allowed for a reduction in latency and network bandwidth requirements, as data is processed and analyzed in the local edge device. As a result, this can result in a faster response time, which is critical for applications that require real-time data processing, such as Smart Farming. Furthermore, this model also yields for a reduction in cost associated with data transfer, since the entire data does not have to be offloaded to the cloud.

Precise Agriculture is a subfield within Smart Farming. It is a data-driven approach to optimize farming practices and increase yields while minimizing waste in resources. Edge computing devices are able to capture important data such as irrigation, fertilization, water level, pest control through multiple sensors deployed in the field. This data can then be very useful for farmers to make inform decisions about their crops.

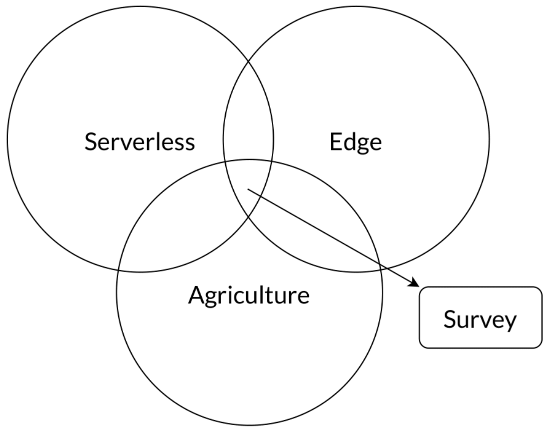

Figure 1: Multi field survey: Serverless, Edge and Agriculture.

However, conditions for edge devices deployed in agricultural fields are generally bounded to low network connectivity or limited power capacity. Smaller devices do not have the same computational capacity, nor the same connectivity, than in a large scale cloud infrastructure. Furthermore, these devices are usually battery constraint, which means that they can only run for a certain amount of time before replacing their power. Given these requirements, efficient computational mechanisms have to be considered when deploying edge devices in agricultural fields.

Serverless computing is an efficient computational model that fits the edge due to its lightweight execution environment. In general, serverless allows applications to be developed and deployed without the need to manage any servers or infrastructure. Furthermore, serverless can automatically scale resources up and down depending on the current demand of the application. Lastly, serverless functions are designed to event-driven. All of these benefits pose a great opportunity for edge devices, allowing a more efficient resource utilization during computation.

Edge devices have very limited capacity, specially in open agricultural fields. Therefore, the question becomes: ***How can serverless computing bring its benefits to smaller devices deployed in precise agriculture?*** As a result, the purpose of this survey is to first evaluate the different optimizations that serverless may have at the edge, and second to study a use case in precise agriculture Figure 1. Therefore, the following are the important questions that should be considered throughout the remaining of the survey:

- *"What are the emerging topics of serverless and edge?"*
- *"What type of serverless applications can run at the edge?"*
- *"What are the opportunities for open research in both areas?"*
- *"How can serverless at the edge impact precise agriculture?"*

Surveys are great ways for answering such questions, as they explore the landscape of the current research to find state-of-the-art solutions. In such alignment, the objective of this survey would then be to present the different research areas that showcase the impact of serverless in compute continuum. Therefore, the main contributions of this survey could be outlined as followed:

(1) **Conducting systematic survey.** Explain the systematic methodology used for conducting the survey by gathering and collecting research papers from serverless and edge computing. The methods used were a combination of two approaches. *(A)* Finding the current trend for open issues, and then selecting the appropriate keywords. *(B)* Constructing queries based on keywords, and using them on Google Scholar[1] and Article Information Parser (AIP)[2], *(C)* Extracting key information from the results using a script *(D)* Building a taxonomy to illustrate the main open research discussions. This is discussed in further detail on section 4.

(2) **Finding serverless optimizations at the edge.** The aim of this contribution is to provide state-of-the-art implementation and frameworks, that would enhance the execution and deployment of serverless at the edge. The goal is to explore the isolation mechanism and resource management of a serverless function, in order to find the optimizations that should work best in edge devices. Among some of the points are cold starts, resource utilization, distribution of workload and energy efficiency. This is touched upon in section 6, section 7 and section 8.

(3) **Showcasing precise agriculture use case.** The goal of this contribution is to provide a clear picture of how serverless at the edge could be applied in a real use case. Understanding both of these concepts in a practical sense is key to grasp the main benefits. As a result, the field of precise agriculture is studied as a use case. This has become the industry that have invested more in sensors for automating manual processes. Therefore, there is still a lot of open research to explore in this area, where efficient computation can be carried through serverless at the edge. This is specifically discussed in section 9.

As per the rest of the survey, the structure will be presented as follows. In section 2, **Related Survey** covers some of the main surveys that are within the field of serverless, edge computing and smart farming. Finally, the **Conclusion** of the survey is presented in section 10, providing a basis for future direction and research.

[1]https://scholar.google.com
[2]https://github.com/atlarge-research/AIP

Surveys	Serverless	Edge	Agriculture
Li et al. [40]	X		
Shafiei et al. [57]	X		
Leitner et al. [39]	X		
Baldini et al. [9]	X		
Xie et al. [69]	X	X	
Aslanpour et al. [5]	X	X	
Khanna and Kaur [33]		X	X
Cong et al. [17]		X	
Liu et al. [42]		X	
Mao et al. [45]		X	
Shi et al. [59]		X	
This survey	X	X	X

Table 1: Related surveys in the field.

2 Related Surveys

Serverless and edge have been studied extensively in the past through multiple literature surveys. Similarly, edge applications in Precise Agriculture have also been studied to a great degree. However, there has not been much work correlating these fields. As a result, the main difference from this survey is the further analysis on how serverless at the edge in agricultural.

A total number of ten surveys were collected during this phase. Table 1 presents a list of related work along with three categories, mainly associated with Serverless, Edge and Agriculture. It is worth pointing out that all of this related work fulfill only one or two relevant categories. This survey covers all the of three topics since it incorporates two fields in Computer Science, i.e. Serverless and Edge, and one in Agriculture. Nonetheless, these related surveys are useful for understanding the open research challenges and visions from each of these categories.

2.1 Serverless Surveys

Serverless is a new field, compared to Edge and Agriculture research, that has developed tremendously over the past couple years. Therefore, exploring the following literatures would provide more knowledge in this computing architecture.

Baldini et al. [9] evaluates serverless computing architecture and the different type of workloads that can be used, in particular burst, compute intensive workloads.

Leitner et al. [39] contributes that serverless functions are a great use case for integration with IoT sensor environments due to the event-driven invocation model.

Shafiei et al. [57] brings interesting perspective on the performance analysis by comparing different language runtimes, container runtimes and memory limits in a serverless function.

Li et al. [40] presents architectural patterns that are used in order to address the open limitations of serverless, e.g. cold-starts and short timeouts.

2.2 Edge Surveys

Edge computing is a more mature field, compared to Serverless, that has grown over the years. Therefore, it is beneficial to explore other surveys, as they provide great insight in the open challenges and opportunities in Edge computing.

Shi et al. [59] presents challenges and opportunities in edge computing in regard to programmability, naming, data abstraction, service management, privacy and security.

Mao et al. [45] provides an insight on how device-to-device (D2D) communication could be established as ways of mitigating limited connectivity in an edge network.

Liu et al. [42] explores the different systems and tools for Deep Learning to achieve a lighter and more adaptive machine learning training at the Edge.

Cong et al. [17] dives deeper into computation offloading strategies where it explores partitioning mobile applications into different granularity levels.

2.3 Serverless Edge Surveys

Despite both fields evolving in parallel, there are a great number of literature surveys that have explored the adoption of serverless computing at the edge.

Xie et al. [69] aims to encapsulate some fundamental areas about communication, coordination and scheduling that serverless functions have to consider in a distributed edge environment.

Aslanpour et al. [5] provides an architectural description on how serverless could be integrated in an IoT edge environment, with key challenges and opportunities to explore.

2.4 Edge Agriculture Surveys

These surveys are important in order to understand more on what type of requirements and systems are built using edge computing in agriculture. There have already been a great number of literature surveys that have evaluated such cases.

Khanna and Kaur [33] goes into details on how IoT can be incorporated in Precise Agriculture. Furthermore, it makes references to the different communication protocols that can be found at the edge when deployed in agriculture fields, which is an important consideration to take into account in a distributed edge network with limited internet connectivity.

3 Background

It is key to understand some background information about cloud, edge and serverless in order to grasp the main idea of the survey. In this section, cloud and edge computing are compared and contrasted based on some requirements, mainly Energy Consumption, Resource Utilization, Latency, Throughput and Cost.

These requirements were primarily chosen to analyze where computation should be carried, either at the edge or in the cloud. There may be some processes that are more heavily energy consuming, which would better to be done in the cloud, for example. However, there may be other cases where latency is a key factor, which would then make edge the best viable option.

Moreover, the distinction of these metrics yields an interesting discussion for how serverless could be optimized at the edge. Hence, this section also provides a common knowledge of what serverless is, and how it works.

3.1 Cloud Computing

Cloud is the delivery of computing services, including servers, storage, databases, software and more. The purpose of this is to provide some knowledge in regard to the requirements already mentioned, in order to determine where computation should actually happen. The following metrics present more in detail the general characteristics from cloud computing:

Metrics	Cloud	Edge
Energy	High	Low
Scalability	High	Medium
Latency	High	Low
Cost	Low	Medium
Mobility	Low	High

Table 2: Metric comparison between Edge and Cloud

(1) *Energy Consumption.* Cloud is normally connected to a power grid, which allows more flexibility as there is no immediate concern about powering the infrastructure. The energy consumption can be related to the amount of computational task required in a workload. Higher operations per workload means a higher energy consumption.

(2) *Scalability.* Cloud provides more computational capacity to scale their services either vertically, by using powerful machines in terms of CPU and Memory, or horizontally, by adding more machines in parallel with the same capacity. Therefore, computing resources can be easily provisioned or de-provisioned to match the workload demands.

(3) *Latency.* Cloud generally have a high latency, depending on the location where the request is made. Requests have to travel across the network to the services in the datacenter, which usually implies a delay in communication. In turns, this affects the application performance.

(4) *Cost.* Cloud adopts a pay-as-you-go pricing model. This provides a great flexibility when dealing with scale, as there is no need to make some upfront investment for resources in advance. This can make cloud computing more cost-effective, particularly with variable or unpredictable workloads.

(5) *Mobility.* Cloud is generally designed to provide centralized access to resources and data. This means that failure on this centralized access can potentially impact the availability of these resources. As a result, cloud does not the capabilities for location awareness.

3.2 Edge Computing

Edge computing is a distributed computing paradigm that involves processing data and performing computational tasks at or near the source where the request is originated As discussed previously, the purpose of this section is to provide some general knowledge on these requirements, in order to determine where computation should actually happen. The following metrics present more in detail the general characteristics from edge computing:

(1) *Energy.* Edge computing devices are generally constraint by their short battery lifespan, which means that they can execute certain amount of processing workloads. Unlike the cloud, they are not particularly designed to carry heavy

computation. However, they are designed for different kinds of applications, such as smart homes.

(2) *Scalability.* Edge computing devices tend to have a limited amount of computational capacity, which makes it more difficult to handle scalability. As a result, they are not well-suited for workloads that are highly variable and unpredictable, since it is more difficult to easily add devices in order to scale up and down the computing resources.

(3) *Latency.* Edge computing devices have a lower latency since computation is happening closer to where the data is generated. This can significantly reduce the delay time for sending and receiving requests. As a result, it can be specially important for applications that require real-time or near-real-time responses, such as autonomous cars.

(4) *Cost.* While edge computing devices tend to be relatively affordable, depending on their computational capacity, it may require some upfront investment to be made, specially when dealing with multi sensor and device deployments.

(5) *Mobility.* Edge computing devices tend to have a higher mobility in terms of location awareness. Due to the decentralized capabilities across multiple nodes, edge computing can provide lower latency and higher bandwidth for real-time processing applications.

Table 2 compares the different metrics previously mentioned in both cloud and edge. The goal with this is to provide a visual reference on how the cloud compares to the edge, specifically when adapting to serverless computing.

3.3 Serverless Computing

Serverless is a computing architecture first introduced by Amazon in 2014 with AWS Lambda. It allows developers to build and run applications without having to manage the underlying infrastructure. The management of the infrastructure is left to the platform for which serverless is running, which is responsible for automatically allocating resources as needed, and dynamically adjusting the scale depending on the workload. As a result, developers only have to focus on writing and deploying code.

The basic entities for serverless computing are functions. Function as a Service (FaaS) is a serverless model in charge of executing code inside a function, which can be invoked either by an event, e.g. user HTTP requests [58], or a message queue. Functions are then scheduled to available computation nodes, where they get executed based on resource availability. Once the execution has ended, these resources get released or reused for subsequent invocations.

Despite the many benefits from serverless, there are also some shortcomings that need to be evaluated for the main discussion serverless edge. *Cold-starts* is one of the most notable issues from serverless. It is the delay that occurs when a serverless function is invoked for the first time after a period of inactivity. This delay is caused by the need to initialize the necessary resources before the function can start executing [19].

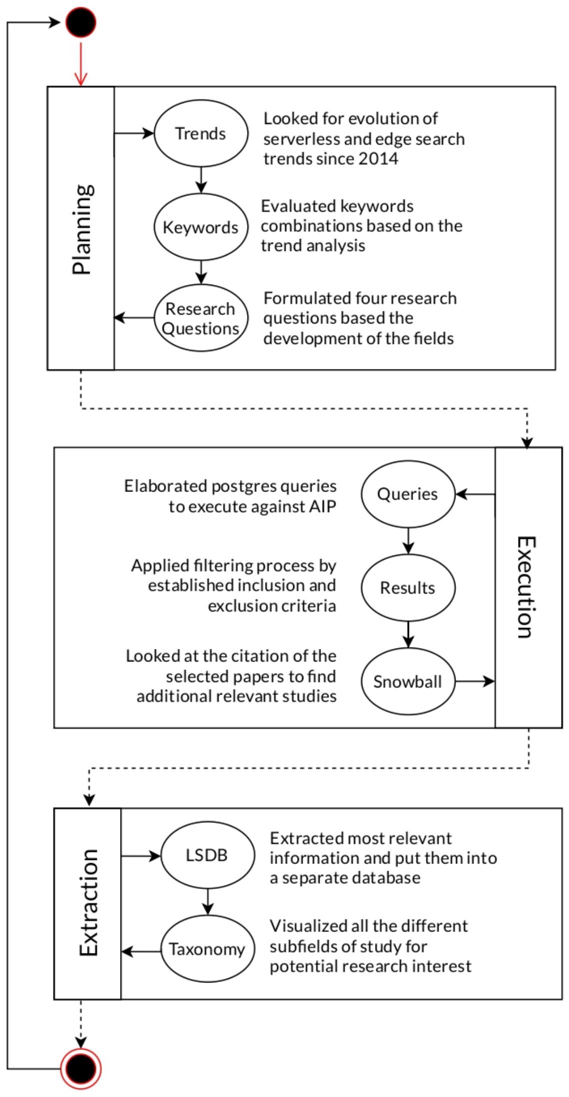
Planning
Trends
Looked for evolution of serverless and edge search trends since 2014
Keywords
Evaluated keywords combinations based on the trend analysis
Research Questions
Formulated four research questions based the development of the fields
Execution
Queries
Elaborated postgres queries to execute against AIP
Results
Applied filtering process by established inclusion and exclusion criteria
Snowball
Looked at the citation of the selected papers to find additional relevant studies
Extraction
LSDB
Extracted most relevant information and put them into a separate database
Taxonomy
Visualized all the different subfields of study for potential research interest

Figure 2: Study Design Detail.

4 Study Design

The design of a survey is an important step to clearly specify how the process of gathering related literature is conducted. Hence, the main objectives of this section are: (1) to convey the process for relating these three different fields, i.e. serverless, edge and agriculture, (2) to present the process for collecting relevant contributions, and (3) to provide an insightful taxonomy that shows the areas of main discussion.

Figure 2 illustrates how the conduction process was carried. **Planning** involved finding keywords and formulating research questions. **Execution** entailed building queries and gathering the results from AIP database. **Extraction** involved collecting the relevant details from literatures and organizing the data in a tree-like data structure.

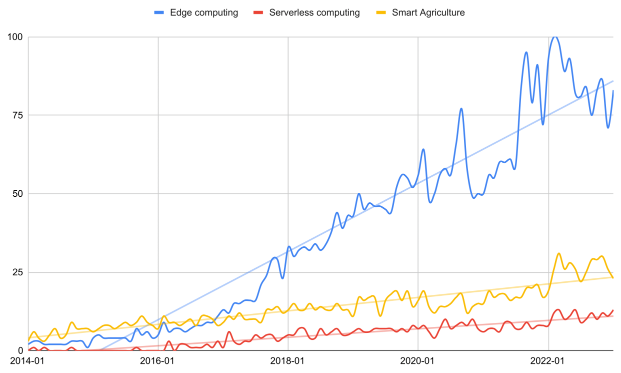

Figure 3: Current search trend for edge and serverless.

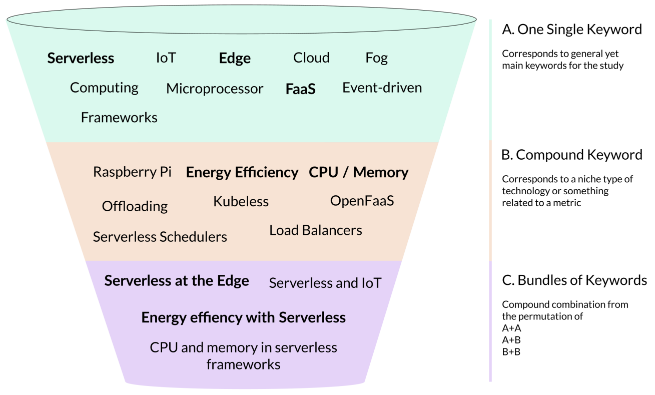

Figure 4: Funnel of keywords.

4.1 Planning

This section covers the initial steps for collecting research papers about serverless, edge and agriculture. On a high level, this process is done by looking at the current google search trends, creating a funnel of keywords, and specifying the research questions.

4.1.1 Trend

Analyzing the current trend in the field is essential for getting a first insight on possible keyword combination. Therefore, *Google Trends* [3] was the tool used to perform such analysis, since it provided useful information about the timeline distribution of a specific keyword.

Figure 3 shows the search trend have developed in these fields (serverless, edge and agriculture) since 2014. Serverless is a recent developing field (introduced in 2014), as compared to edge. As a result, this survey review is evaluated based on this timeline.

Lastly, Google Trends provides a good set of combination of key terms that can be used for search engines. Hence, they will be used in the next stage of Keyword Analysis.

4.1.2 Keyword Analysis

Based on the collection of keywords gathered from Google Trends, three keyword tiers are presented in Figure 4 to show the different categories and description.

One Single Keyword (High level). This category of keywords is used for searches that are mostly generic. These keywords are helpful because they aggregate the first papers or literatures that exist in the field. For example, they would be extremely useful in the process of collecting related surveys.

Compound Keywords (Somewhat specific). This category corresponds to a more specific set of keywords to search contributions in literatures. For example, a combination of more than one word is used to find interesting results about serverless load balancing mechanisms.

Bundle of Keywords (Specific). This category dives deeper into the combination of keywords, yielding the results for most relevant/seed papers. These search terms usually involved more extensive keyword combination, e.g. "serverless at the edge" or "energy efficiency in serverless in IoT."

[3]https://trends.google.com/trends/?geo=NL

4.1.3 Research Questions

The most important step for the planning phase of the design is drafting research questions that would ultimately reflect the main contributions of this survey [32].

These research questions take into account two important requirements necessary for edge computation in agriculture fields.

- *Energy consumption:* serverless computation deployed on edge devices has to be efficient in processing data, as this would improve battery life of the device.
- *Performance:* how quickly and accurately data can be processed and analyzed in real-time, with minimal latency and bandwidth requirements.

There are two major research questions that correlates serverless, edge and agriculture. From these, there are sublevel questions that can be extracted. However, the sublevel questions ought to be discussed in later sections.

RQ1

What kind of optimizations can be found in terms of energy consumption and performance (CPU and memory) from serverless at the edge?

As previously discussed, there are some benefits that serverless brings to edge computing devices. The point of this question is to find solutions that could optimize computational execution on smaller devices.

RQ2

How can serverless at the edge be implemented in the precise agriculture field?

Since this survey incorporates a real use case study for how serverless and edge could be applied in agriculture, it is essential to also research relevant implementations done already. Therefore, the main objective of this research question is to study the landscape on how the agricultural field has changed with IoT edge devices, specifically finding implementations using serverless computing.

Query 1: `SELECT * FROM publications WHERE (title ILIKE '%serverless%') AND (title ILIKE '%edge%')`

Query 2: `SELECT * FROM publications WHERE (title ILIKE '%serverless%') AND (title ILIKE '%edge%') AND (title ILIKE '%agriculture%')`

4.2 Execution

Once the trend has been analyzed, and research questions been made, it comes the execution of the survey. On a high level, this process is done to building the queries, making selection criteria, and snowballing references.

4.2.1 Queries

There were two data sources used during the process, AIP and Google Scholar. AIP was the only database where literatures could get extracted using SQL queries, since this is a Postgres database. Therefore, two fundamental queries were initially made.

Q1- Serverless at the Edge. This query aims at retrieving research papers that are more related to state-of-art implementations or algorithms that can be applied for serverless functions to execute efficiently at the edge.

Q2- Serverless at the Edge: Precise Agriculture. The goal of this query is to find solutions for how serverless and edge work in precise agriculture. Since this is a developing area, there could be very limiting number of literatures. However, the general idea is to get a field for how the industry is integrating sensing devices.

4.2.2 Selection criteria

The selection criteria of this survey can be based on an inclusion set of requirements that need to be taken into account when narrowing the search. **??** provides an overview and rationale of these evaluated criteria, which could also be summarized as:

I1- Year. The year of publication is a consideration to make, specially when discussing serverless as a newer field compared to the other two, edge and agriculture. Therefore, it is most relevant to search for literatures from 2014.

I2- Database searches. The search engine is an important tool when evaluating paper inclusion. Literatures have to be discoverable in both engines in order to be considered in the survey.

I3- Research questions. Lastly, research questions are key factors when taking into account the inclusion of a research paper. Papers were evaluated mostly on Title and Abstract, looking for important terms and definitions that would provide more insightful information about the development of these fields.

4.2.3 Snowball

This type of the systematic search involved using the reference list of a paper or the citations to the paper in order to identify additional literature [67] that may be relevant to the fields. The process involved going through the references from literatures found with the initial queries. While doing so, interesting implementations were found and needed to be considered accordingly. Hence, this process permits the flexible expansion of the systematic search, since it explores other literatures based on the citations rather than a query.

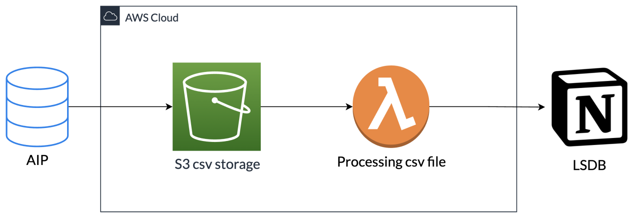

Figure 5: Data extraction from AIP to LSDB.

Bibliography data	Basic information about the literature
Author	The list of authors
Title	The title of the paper
Published Year	The data the paper became available
Number of Citations	The total number of citations
Categorization	Specify more information about a paper
Tagging	Assign a label to papers
Classification	Rate paper based on relevance to topic
Indicator	Identify seed papers by assigning icons
Findings	Details on the proposed solutions
Main Contributions	Summary of the paper

Table 3: Data extraction properties

4.3 Extraction

In this phase of the study design process, important information about the selected papers is extracted with the purpose of condensing and synthesizing everything in one location. On a high level, this phase is done by using a custom database in Notion[4] , and by building a taxonomy for main topic discussions.

4.3.1 LSDB

This survey presents *Literature Studies Database (LSDB)*, which is a Notion database used for storing all the extracted information from AIP and Google Scholar about relevant papers in the literature.

Since the extraction process consisted of LSDB, an automated pipeline was created in order to populate this database with the literature findings. Figure 5 represents such pipeline, which consists of the following components:

- *AIP queries:* are saved in a csv format. Then these files are pushed to AWS cloud for integration with Notion.
- *AWS S3[5]:* is used as a storage mechanism of the csv files.

[4]https://www.notion.so/
[5]https://aws.amazon.com/s3/

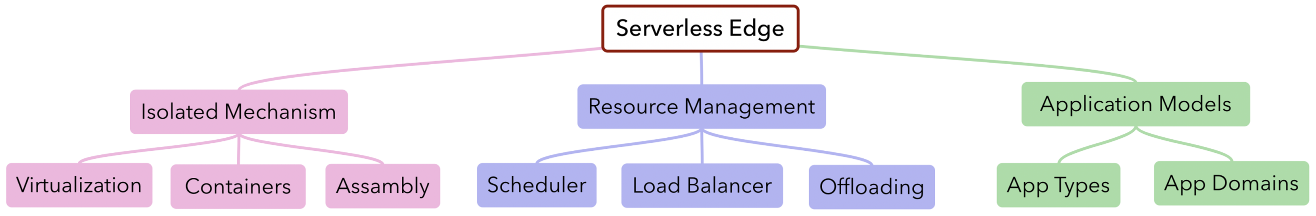

Figure 6: Serverless Edge Taxonomy.

- *AWS Lambda:* gets triggered every time a csv file is put into the bucket. This function preprocesses the csv file and make HTTP calls to Notion API.

During the extraction process, there are groups of properties that are mainly considered for each of the papers.

- *Bibliographical Information.* This refers to details such as Authors, Title, Published Year, Number of Citations.
- *Categorization.* Once information have been collected, there are three main categories that would apply to each paper. **Field** which refers to a label assigned to a paper based on its field. **Category** which refers to the general domain of discussion within the field. **Implementation** which refers to the methodologies used in the literature.
- *Findings.* Some high level information about the main contributions are recorded, which would then be useful to quickly recognize what the paper is about.

4.3.2 Taxonomy

The final step in the extraction process corresponds to building a taxonomy with the relevant information gathered from the different literatures studied. As a result, taxonomies are a great way to understand better the different subfields in a literature, since they provide a structured and detail decomposition of certain concepts [65]. Decomposition are great to look at an overview of the possible and attempted areas that have researched. Using this overview, researchers help get guided through the process for finding feasible solution for these challenges. Hence, the taxonomy presented Figure 6 makes it more visual the main points for discussion in the survey.

5 Serverless Anatomy

Before diving deeper into the analysis of serverless at the edge, it is important to have a prior knowledge of how a serverless function is structured. Therefore, considering the limitations from serverless, mainly the cold-start issues section 3, this section provides an evaluation on what specific components need to be optimized for efficient computation at the edge. This subsection covers two areas in serverless computing, the architecture and the lifecycle of a function.

5.1 Serverless Architecture

A serverless function is generally composed by four different layers, as depicted in Figure 7. These layers can be defined as Virtualization, Encapsulation, System Orchestration and System Coordination.

Virtualization Layer. Even though functions are meant to be serverless, there is still a need for an environment in which they have to be executed. As a result, this layer provides a secure and isolated environment for functions to run on top of a virtual machine (VM). The purpose of this VM is to provide the necessary runtime environment, dependencies, and system libraries required by the application to execute.

Encapsulation Layer. This layer deals with customized triggers and executions, as well as provides data metrics collection for communication and monitoring. Pre-warm pools are important to avoid any bottleneck of resources in this layer. The aim of this layer is to dynamically pre-installing requirements for runtime execution.

System Orchestration. The orchestrator is the layer that binds everything together. It achieves scheduling optimizations at three different levels: resource, instance and application. Furthermore, they are in charge of maintaining a high availability and stability of the function, by dynamically adjusting the load as the computational workload changes.

System Coordination. The coordination provides the necessary components for a function to integrate a series of services. For example, **Backend-as-a-Service (BaaS)** is a term that refers to APIs and SDKs that can be deployed with the function. These services provide storage, queuing services, trigger binding, among other components necessary for the lifecycle of a function.

5.2 Serverless Lifecycle

The lifecycle of a serverless function is a fundamental concept to understand when putting it in context of edge. This will help in identifying which individual stages can be optimized for smaller computational devices.

There are multiple stages in which a serverless function goes through its execution lifecycle. Figure 8 provides a more visual representation of such a process, depicting how the execution environment of a Lambda function evolves over time [68].

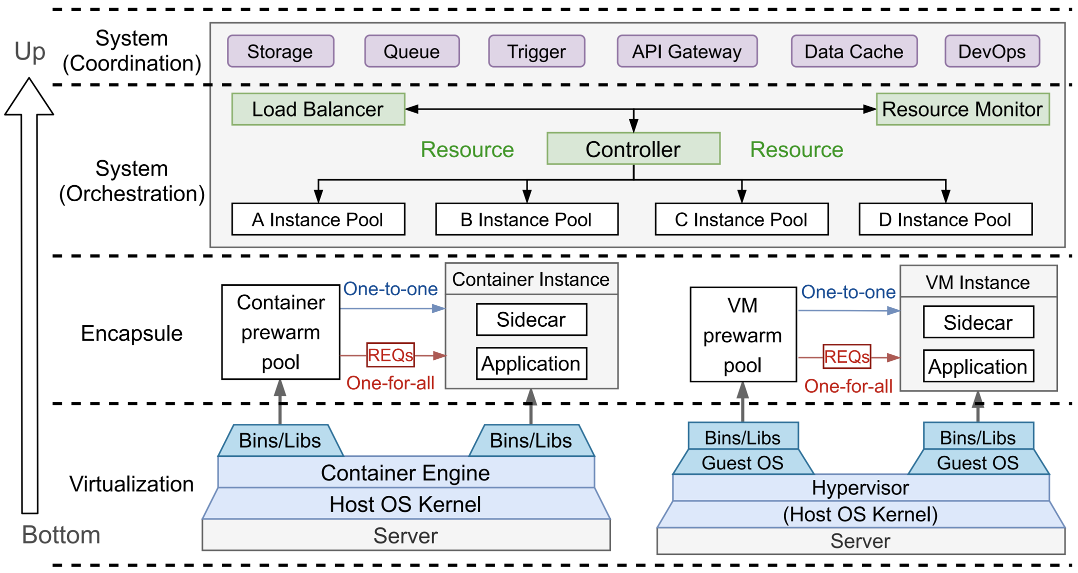

Figure 7: Serverless Architecture [40].

Init Phase. During this phase, there are three important tasks to consider in the lifecycle: **(1)** Start all extensions, **(2)** Bootstrap the runtime, **(3)** Run the function's static code. The Init phase ends when the runtime and all extensions signal that they are ready by sending a Next API request.

Invocation. When a Lambda function is invoked in response to a Next API request, Lambda sends an Invoke event to the runtime and to each extension.

Shut Down. When Lambda is about to shut down the runtime, it sends a Shutdown event to each registered external extension. Extensions can use this time for final cleanup tasks. The Shutdown event is a response to a Next API request.

6 Isolation Mechanisms

There are several isolation mechanisms implementations that have been developed in serverless functions. This subsection first presents an overview from all of these mechanisms, and second identifies which of these mechanisms would be most suitable for the edge, based on its requirements. As discussed in section 4, **RQ1** breaks down into multiple sub research questions. One of them is the following:

RQ1.1

What kind of isolation mechanisms in serverless provide a better energy consumption and performance results to edge devices?

When discussing the adoption from serverless at the edge, it is important to consider the efficiency in terms of performance and energy consumption from each of these techniques, as they would determine whether it is possible to deploy in smaller devices with certain constraint capabilities, such as limiting battery and compute capacity.

Efficiency in other words can refer to the startup latency the function has. In particular, after a function performs the computation, there needs to be some ways of recycling the environment so that subsequent functions can still execute without having to wait to start all over again.

As a result, there are some isolation mechanism that allow for this process to be lot faster. Figure 9 provides a visual representation through a taxonomy, showing the different components that are discussed in this subsection.

6.1 QEMU

Traditional virtualization mechanism have been applied to serverless underlying infrastructure since the inception of FaaS as a computational model for executing code. Even though this mechanism provides a high level of isolation and flexibility, it is the least efficient, from a performance perspective, because it has a very high startup latency. Part of the reason is that VMs have high resource overhead, which means high memory and storage footprint. On the other hand, serverless functions are more suitable for a lighter weight execution environments, since they are designed for quick invocation and compute processing. Therefore, applying traditional virtualization to a serverless environment would not be ideal at the edge since this would imply higher execution time, which would impact performance and energy consumption.

6.2 MicroVMs

MicroVM is a slimmed version of a virtual machine. It provides a minimal, stripped down environment for single purpose applications and workloads. Therefore, it is designed to be lightweight, fast and secure, with a focus on delivering high performance with low overhead. Compared to a traditional VM, it consumes less amount

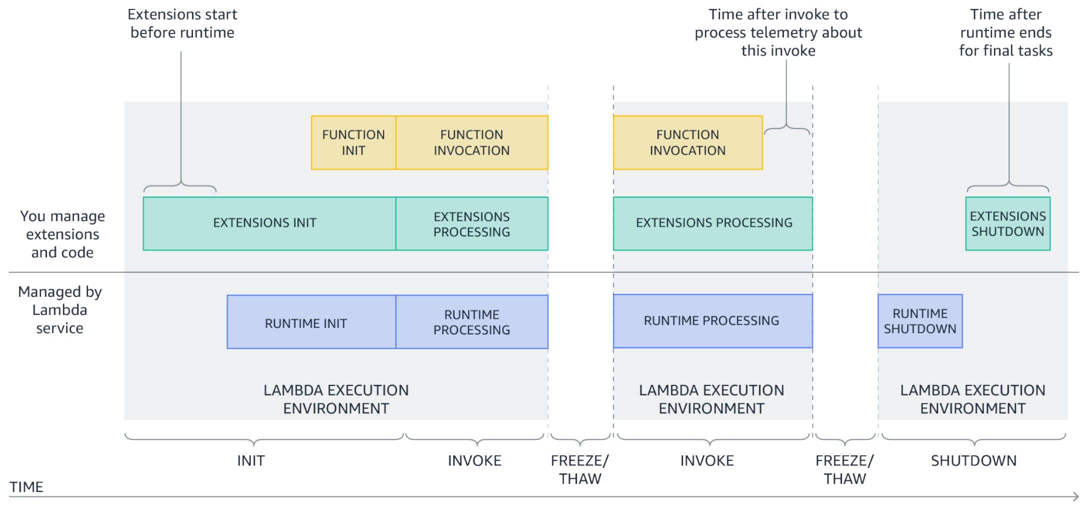
Extensions start before runtime
Time after invoke to process telemetry about this invoke
Time after runtime ends for final tasks
FUNCTION INIT
FUNCTION INVOCATION
FUNCTION INVOCATION
You manage extensions and code
EXTENSIONS INIT
EXTENSIONS PROCESSING
EXTENSIONS PROCESSING
EXTENSIONS SHUTDOWN
Managed by Lambda service
RUNTIME INIT
RUNTIME PROCESSING
RUNTIME PROCESSING
RUNTIME SHUTDOWN
LAMBDA EXECUTION ENVIRONMENT
LAMBDA EXECUTION ENVIRONMENT
LAMBDA EXECUTION ENVIRONMENT
INIT
INVOKE
FREEZE/ THAW
INVOKE
FREEZE/ THAW
SHUTDOWN
TIME

Figure 8: Serverless Function Lifecycle [68].

of resources, which makes it susceptible for smaller devices that have constraint capacity. Unlike containers, it provides a higher level of security by separating the use of the kernel per serverless function. In other words, each function runs its own microVMs, which ultimately brings isolation benefits. At the moment, there are two major microVMs that analyzed for performance optimizations in serverless functions.

FireCracker [12] is an open source, lightweight virtualization technology which powers AWS Lambda. It works by leveraging Kernel-based Virtual Machine (KVM) hypervisor to create small virtual machines, running a stripped down Linux kernel with only the necessary drivers loaded [47]. Furthermore, it provides a more secure environment by adopting a VM Monitor (VMM) based on KVM [54].

gVisor [38] sandbox that helps provide secure isolation for containers, while being more lightweight than a virtual machine (VM). The core of gVisor is providing a Guest kernel that runs as a normal, unprivileged process that supports most Linux system calls. In other words, the idea is that each individual serverless function is run on its own VM, completely isolating the main functionalities from the underlying Host kernel.

These approaches work particularly well in cloud environments, where performance at scale can be defined by the hardware capacity from the physical machine. However, similar to containers, microVMs are not suitable for all type of edge devices, specially those with a lower battery and computational constraint. Therefore, other methods have to be explored when considering the abstraction layer that serverless should adopt at the edge.

6.3 Unikernels

Unikernel is a single-address-space machine image with a minimal set of OS services compiled specifically to run only a single application [63]. This approach is a natural fit for the FaaS model because of the lightweight and strong security isolation. Unikernel provides better performance to serverless functions by only supporting the necessary libraries and drivers that functions actually need [29]. Furthermore, it typically uses a single kernel for both the guest and host operating system, which allows for a small memory and disk footprint. There are multiple open source projects that have been studied to optimize performance of serverless functions. In general, these can be grouped in two categories: language-based and POSIX-based [49].

- *Mirage OS* [6] is a language-based library operating system that produces unikernels by compiling and linking OCaml code into a bootable Xen VM image [43].
- *OSv* [7] is an POSIX-based unikernel that provides a new guest operating system designed specifically for running a single application on a virtual machine [35].

The difference is that language-based unikernels offers higher performance but with less compatibility with conventional systems, while POSIX-based allows more flexibility in the application development, with the cost of a larger memory footprint.

Unikernels are specially designed for running in smaller IoT devices. For this reason, finding performance optimizations for serverless functions using Unikernels should be the priority of the discussion. As a result, there has been some research development in serverless platforms for edge devices.

UniFaaS [49] is a prototype edge-serverless platform which leverages unikernels tiny library single-address-space operating systems that only contain the parts of the OS needed to run a given application to execute functions. The result is a serverless platform with extremely low memory and CPU footprints, and excellent performance. It has been designed to be deployed on low-powered single-board computer devices, such as Raspberry Pi or Arduino, without compromising on performance.

[6]https://mirage.io/
[7]http://osv.io/

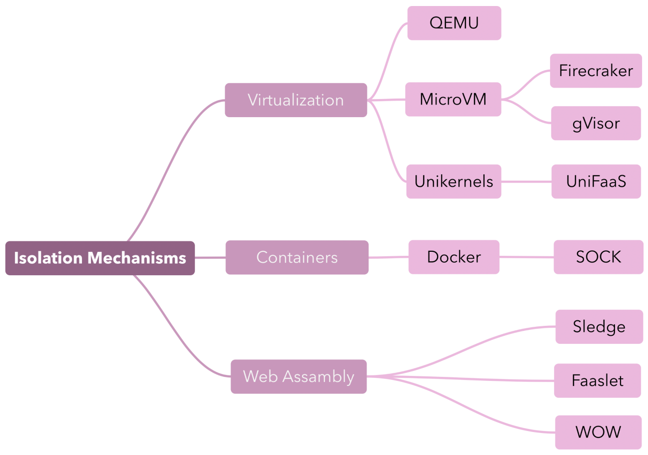

Figure 9: Isolation Mechanism Taxonomy.

6.4 Containerization

Container is another option for resource isolation in a serverless environment. It is a lightweight alternative to traditional Hypervisor-based virtualization. It implements virtualization at the OS level, instead of at the hardware level. Furthermore, containers are able to achieve isolation through the use of cgroups and namespace, which allow sharing the same OS kernel while providing a level of abstractions between individual processes. Docker is one of the most used container engine for packaging and deploying applications. It provides RunC container runtime, which sets up the environment with the respective libraries and system tools for code to execute in isolated processes. However, further optimizations have been explored to make this container runtime more efficient in a serverless environment.

SOCK [53] is a lean container proposed for serverless functions. It provides an even lighter runtime environment since it removes a lot of the redundant mechanism imposed by Docker containers. Hence, this ends up being a great solution for functions that need to run more efficiently in startup latency and throughput, two requirements that directly translate to performance optimization.

A containerized serverless environment could be deployed at the edge in smaller devices. However, this may not be suitable for all cases of edge devices, because containers run at the OS level, which means that their image would occupy some space in memory. As a result, there are other solutions that provide a lower cold-start latency, with the same performance level.

6.5 WebAssembly

WebAssembly is an open web standard for executing portable binary-code in the underlying host environment. Although mostly used for web application purposes, the specification for WebAssembly has been adapted to work outside browsers, specially in a serverless environment. The difference with this isolation approach is that it completely bypasses the Linux kernel abstraction, optimizing the performance and latency of a function execution time. To this extent, there has been some research development, with the goal of finding the most optimal execution time for a serverless function.

Isolation	Strategy	Startup Latency
QEMU	Virtualization	>1000ms
Docker	Containerization	50ms - 500ms
FireCracker [12]	MicroVMs	50ms - 500ms
gVisor [38]	MicroVMs	50ms - 500ms
SOCK [53]	Containerization	10ms - 50ms
UniFaaS [49]	Unikernel	10ms - 50ms
Sledge [26]	WebAssembly	8ms - 25ms
Faaslet [60]	WebAssembly	5ms - 10ms

Table 4: Isolation Mechanism for Serverless at the Edge

Sledge [26] is a low-latency serverless execution at the Edge. It enables a light-weight function instantiation and isolation environment which leverages kernel bypass to enable specialized serverless function scheduling. The runtime focuses on efficiency of serverless functions, and enabling strong spatial and temporal isolation of multi-tenant function executions.

Faaslet [60] is an isolation mechanism for data-intensive serverless computing. It guarantees strong memory and resource isolation. Furthermore, it supports stateful functions with efficient shared memory access, and are executed by Faasm, a distributed serverless runtime. To deal with cold-starts, Faasm runtime pre-initializes an environment, yielding for a more efficient execution time.

WOW [25] is a prototype for executing Wasm workloads in Apache OpenWhisk. It is extensible in terms of the Wasm runtimes it supports, requires minimal modifications to OpenWhisk, and seamlessly integrates Wasm special features such as capability-based access control.

6.6 Comparison of Isolation Mechanims

In the discussion for adapting serverless at the edge, not all of these virtualization mechanisms are going to be well suited in smaller devices. It is crucial to consider that the requirements on edge devices are totally different from the cloud, namely due to the constraint of capacity and resources available. Optimizing for lower response time, while maximizing on computing efficiency, requires the right isolation mechanism in the architecture of a serverless function. As a result, Table 4 presents a comparison of the previously discussed isolation frameworks and strategies, with their corresponding startup latency. From this, there is a noticeable improvement in startup latency when serverless functions run in either **Unikernels** or **WebAssembly**.

7 Resource Management

Resource management is another important layer that should be discussed when trying to optimize serverless at the edge. Due to some of the limitations already mentioned from edge computing, there needs to be an alignment to find a suitable set of implementation or algorithms in serverless. As discussed in section 4, **RQ1** breaks down into multiple sub research questions. The second one is as follows:

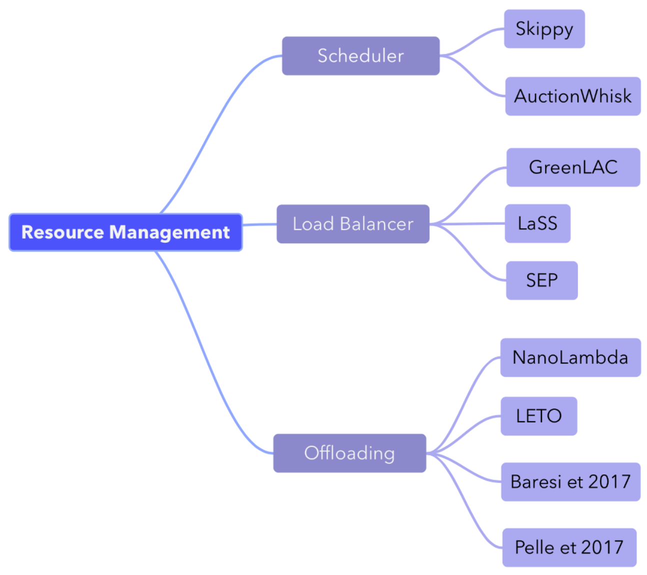

Figure 10: Resource Management Taxonomy.

RQ1.2

What kind of resource management components in serverless provide a better energy consumption and performance results to edge devices?

This subsection provides an overview of what components are necessary to adapt in the resource manager from a serverless function so that it can run efficiently in smaller devices. Figure 10 provides a taxonomy which presents the breakdown of these components.

7.1 Resource Scheduler

On a high level, a resource scheduler is the component that manages the allocation, provisioning and scheduling of resources of a system. The main goal is to ensure that all the running processes get the necessary resources to run efficiently, while maximizing system utilization. As a result, there are algorithms and heuristics to determine the best way to allocate resources to tasks.

Resource scheduler becomes a crucial component in edge computing devices due to their limited capacity. Due to the constraints present in smaller devices, schedulers have to be designed to efficiently manage the underlying resources of a smaller device, such as CPU, memory and storage, while also balancing the performance of process execution.

7.1.1 Components of Resource Scheduler at the Edge

Serverless functions can play a crucial role in maximizing the performance of a scheduler, while minimizing the energy consumption. However, it would be important to understand the main parts of edge schedulers in order to identify in greater detail where serverless could bring benefit to the edge environment.

(1) *Resource Monitor.* is responsible for collecting data on the current state of resources available at the edge, such as CPU utilization, memory usage, network traffic and other performance metrics.

(2) *Resource Allocation.* is responsible for deciding how to allocate resources to different services running on a device.

(3) *Dynamic Provisioning.* is responsible for moving tasks between different processing elements, with the ability to increase or decrease the resource allocation.

(4) *Policy Manager.* is responsible for enforcing policies and constraints on resource allocation and scheduling, in order to avoid resource exhaustion from an individual process.

(5) *Optimization Engine.* is responsible for determining the most optimal way to allocate and schedule resource based on current state of the edge device.

7.1.2 Limitations from Resource Scheduler at the Edge

In addition to scheduler components explained, there are also certain challenges that arise at the edge. By presenting the following limitations, there is a possibility to find certain gaps where serverless functions could bring a positive value to the edge computing environment.

(1) *Dynamic environments.* edge devices are often deployed in remote or unpredictable locations. Therefore, a scheduler must be able to adapt to these changes in order to adjust the allocation of resources in real-time.

(2) *Energy consumption.* edge devices are usually bounded by a battery lifetime. As a result, a scheduler must be designed to minimize the energy consumption while ensuring the optimal computational capacity.

(3) *Heterogeneity.* edge devices have different hardware and software configurations. This means that a scheduler at the edge need to be able to allocate computation to a wide range of devices.

7.1.3 Serverless Resource Scheduler

Resource management in a serverless environment refers to managing the resource requirements of an application workload, in other words spawning of the sandbox environment, allocating the execution environment and the runtime environment. The tasks carried by the scheduler can be related to the lifecycle of a function (subsection 5.2).

Pigeon [41] creates a function-oriented Serverless framework by introducing an independent and finer-grained function-level resource scheduler on top of Kubernetes[8]. The goal of this framework is to minimize cold-starts (section 3) by implementing the scheduling functionalities from Kubernetes, but adapted to a serverless environment. With function level resource scheduling, serverless functions can be directly dispatched to pre-warmed containers, which greatly reduces the resource overhead.

FnSched [62] is a function scheduler designed to minimize provider expenses while providing acceptable request latencies. FnSched enables autoscaling by elastically adding and removing functions, based on the incoming workload. FnSched scheduler

[8]https://kubernetes.io/

algorithm determines the number of CPU-share from active containers. In addition, it monitors the latency of applications in order to increase the number of CPU-share capacity.

7.1.4 Edge Serverless Schedulers

When deploying FaaS at the edge, it is important to consider that scheduling decisions from serverless platforms. This decision will depend on the workloads, the size of data, data transfer costs over different network paths, and the cost for setup time of resources at each location. As such, the following studies have researched the resource allocation and scheduling mechanisms in serverless computing, with the goal of finding optimizations for edge environments.

Skippy [56] facilitates the efficient placement and allocation of serverless edge functions on distributed and heterogeneous clusters. It leverages the tight integration of the scheduler with a simulation framework, in combination with existing multi-objective optimization algorithms, to improve function execution time, network usage, edge resource utilization, or cloud execution cost.

AuctionWhisk [14] proposed an auction mechanism to allocate resources among multiple fog nodes in a decentralized and self-organizing way. This approach allows for dynamic and efficient allocation of resources to serverless functions, improving both resource utilization and response time. This improves edge computation by considering data location and available resources in real-time to make optimal function placement decisions.

Aslanpour et al. [4] proposed resource scheduling algorithms to place functions on edge nodes powered with battery and renewable energy sources. Their goal was to maximize the operational availability of edge nodes while minimizing the variation thereof without compromising the throughput. By doing so, they applied sticky offloading and warm scheduling to reduce recurrent function invocations.

Studying the different scheduling mechanisms from an edge serverless function platform can provide a greater insight to performance optimizations and the energy consumption necessary to adapt to smaller devices. However, other components also play an important role in providing low latency response times at the edge, among one of is the load balancer.

7.2 Load Balancer

On a high level, a load balancer is a component within a resource manager that is in charge of distributing traffic according to some load balancing algorithms or policies. The main goal is to avoid overloading resources by monitoring incoming traffic and distributing among the available computational capacity. Load balancers at the edge are notoriously important for resource utilization in smaller devices. But before diving deeper into how edge load balancers might work with serverless, it is worth to first note the deployment models that load balancers may have at the edge.

7.2.1 Load Balancers Deployments at the Edge

Knowing where load balancers operate can help determine optimization techniques that can be employed by a serverless function to provide a better performance and quality of service (QoS). Hence, this can be classified into four different areas:

(1) *Content Delivery Networks (CDNs).* These are network of distributed servers or devices that are used to deliver content or data to end-users or devices. Load balancers can be deployed at various locations within the CDN network, such as edge servers, regional points of presence (PoPs), or global PoPs.

(2) *Internet Service Providers (ISPs).* ISPs can deploy load balancers at the edge of their networks to distribute traffic across their network of servers or devices, and to improve performance and reduce latency for their customers.

(3) *Multi-access edge computing (MEC).* MEC is a network architecture that enables computing resources and services to be deployed at the edge of the network, in proximity to the end-users or devices. Load balancers can be deployed within the MEC architecture, to distribute traffic across the network of resources and improve performance.

(4) *Cloud Edge.* Cloud providers may offer load balancers as a service at the edge of their network, to enable customers to distribute traffic across their network of servers or instances, and to improve performance and availability.

Given that different deployments at the edge, it is crucial to study the fundamental requirements that load balancers should consider at the edge.

7.2.2 Requirements of Load Balancers at the Edge

Understanding these requirements can help provide a framework for how serverless functions should operate under such conditions, while providing the performance and energy consumption optimization that is necessary.

(1) *Scalability.* Edge computing is present in billions of devices. Therefore, there is a high requirement to have very scalable load balancing capabilities that can distribute the traffic spikes and accommodate increasing traffic demands.

(2) *Redundancy.* Failure is certainly a something to consider when running systems that scale to billions of devices. Therefore, high availability and redundancy are critical for load balancers at the edge, as incoming workloads first interact with them when coming into the system.

(3) *Geolocation Based Routing*. Edge devices are dispersed across a wide range of locations. Therefore, there is a requirement for load balancers to ensure that traffic is routed to the closest available resource based on the location where the data originated

(4) *Performance.* In order to handle a high volume of traffic, generated from the billion of edge devices across a wide area, it is necessary that load balancers use optimized algorithms and policies that can efficiently distribute traffic with low latency response.

As a result, serverless load balancing mechanisms have to take these requirements into account when distributing the incoming jobs into multiple devices.

7.2.3 Serverless Platform Load Balancers

Load balancers in a FaaS platform are designed to route queries and schedule functions to achieve a high distribution between nodes in a cluster. As a result, serverless inherits some of the load balancing algorithms and strategies that originated from previous virtual execution environments. Among some explained in this survey are: *The Least Connections* and *Locality Aware*. The following literatures provide more of an insight into how these algorithms interrelate in a serverless platform.

PASch [6] proposed a package-aware scheduling approach, where serverless functions are packaged with metadata that describes their resource requirements and performance characteristics, such as amount of CPU and memory. The algorithm proposed would then place the incoming workloads to workers in a serverless functions based on the metadata of these packages. The workloads would get distributed to the least loaded workers available.

FaDO [61] proposed a new architecture for managing and deploying serverless computing environments, implementing multiple load balancing algorithms, The Least Connections and Round Robin. The paper also describes how the FaDO architecture can be used to implement load balancing across multiple edge and cloud clusters to distribute request across multiple workers in a serverless function.

Palette Load Balancing [1] proposed a new load balancing technique that takes advantage of the locality of serverless functions, with the goal of optimizing the distribution of workloads. Furthermore, it suggested that algorithms, such as round-robin and least-connections, do not work well for serverless computing environments because they do not take into account the data locality of the functions, which can be a differentiator in higher latency and lower throughput.

7.2.4 Edge Serverless Load Balancers

So far, the discussion has been mostly on how to optimize load balancers mechanism for serverless functions that work in the cloud. However, the main idea is to see whether these same mechanisms could work at the edge, given the requirements already specified.

GreenLAC [18] proposed a load balancer for serverless platforms to distribute the workload among edge nodes based on resource availability and network conditions, taking into account factors such as CPU usage, memory usage and network latency. This uses an open-source component for AWS Greengrass [9] to achieve load balancing and resource monitoring on edge devices.

LaSS [66] proposed a new approach to addressing the latency issues in edge networks by ensuring fair share resource allocation to avoid overloading the system. It takes a similar balancing approach than *PASch* in such a way that it gathers metadata about the workload, primarily the size, and then distributes the load using a weighted round-robin approach.

SEP [11] addressed the issue of centralized load balancing from serverless functions in cloud environments. They proposed a new architecture for serverless at the edge, consisting of two layers: cloud layer, responsible for abstraction of serverless functions, and a distributed edge layer, responsible for low-level compute and storage of devices. As a result, an external load balancer would be sitting between these layer, in order to allocate workloads evenly.

Load balancing mechanisms is one of the fundamental concepts for distributing workloads based on certain algorithms. However,

[9]https://aws.amazon.com/greengrass/

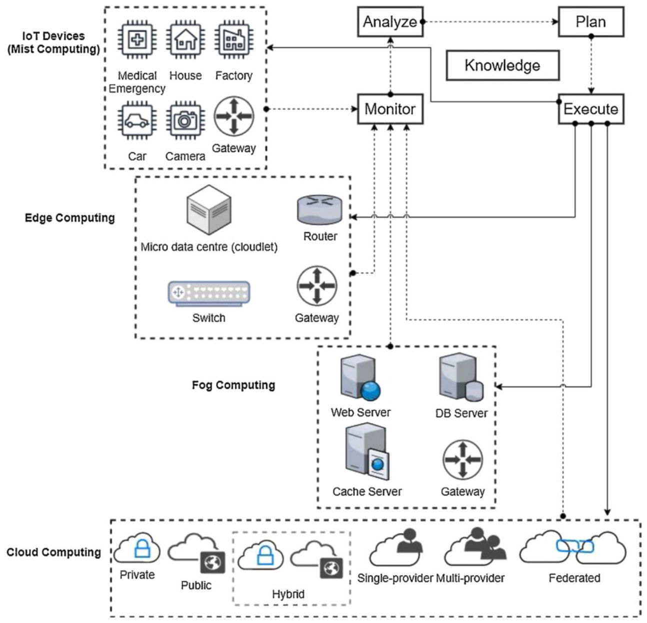

Figure 11: Landscape of Computing Paradigms [3].

there is one final resource component that should be studied in this exploration of serverless edge, that is offloading of computation.

7.3 Offloading

On a high level, offloading can refer to the process where some amount of computational workload is moved from a local system or device, to another in a different location. This term is generally applicable within edge computing, since it plays a big factor in the constraint of smaller devices. For example, if a device is not able to handle the execution of a workload, it becomes more efficient to move this workload into the cloud, where there are more resources available. Therefore, offloading can yield for a better response depending on certain parameters that should be evaluated.

7.3.1 Offloading Computing Models

Computing models are important to understand in order to grasp a better idea of how offloading operates at different levels. Figure 11 provides a visual representation of the different computing models studied in this survey.

(1) *Mist Computing (MC)*. This refers to a network of devices, such as sensors, actuators, work together to process and analyze data locally. From the figure, this Figure 11 could be visualized in the IoT layer, where devices are responsible for sensing and actuating based on triggers. An example of this is a Raspberry-Pi[10].

(2) *Edge Computing (EC)*. This layer is usually where dedicated gateways and routes are located, in order to act as an intermediate between the smaller devices and fog/cloud. The edge can be connected to the cloud for further processing

[10]https://www.raspberrypi.org/

and storage [30]. An example of this is Cloudlets[11], who bring the computation closer to the IoT devices while being.

(3) *Fog Computing (FC).* This is a higher layer, closer to the cloud than to the IoT devices. Furthermore, it offers various cloud services endpoints to offload their computation at low latency [30]. An example of this is a micro-data center with devices closer to the cloud to offload compute capacity.

(4) *Mobile Cloud Computing (MCC).* This layer provides a more persistent and powerful infrastructure which is backed by a higher computational capacity. As a result, it offers a viable option for smaller devices to offload heavier workloads. However, MCC is the farthest option to where the data is generated, yielding a higher latency.

(5) *Multi-access Edge Computing.* Although not present in the Figure 11, this classification is concerned with augmenting cellular and wireless infrastructure with computing capacity to process workload from multiple endpoints connected via 4G or 5G [30].

7.3.2 Offloading Directions

Offloading can occur from the cloud to the edge, or from the edge to the cloud, depending on the specific use case and requirements.

Cloud-to-Edge: involves transferring some of the computing workload from a centralized cloud system to edge devices or gateways located closer to the data source. This can help reduce latency and bandwidth requirements, and enable real-time processing and analysis of data.

Edge-to-Cloud: involves transferring some of the computing workload from edge devices or gateways to a centralized cloud system for further processing, storage, or analysis. This can help leverage the resources and capabilities of the cloud system, such as data analytics tools, machine learning algorithms, or high-performance computing clusters.

Evaluating this tradeoff would depend on the requirements and use case of each individual system. However, this is relevant when discussing the different offloading models that can happen, specially when running serverless functions in a more constraint computational environment.

7.3.3 Task Offloading Mechanisms

As it turns out, there are several implications in offloading which would be useful in implementing serverless functions at the edge. The idea is to first present these mechanisms and then understand how a serverless function would fit into either of these tasks offloading.

(1) *Application-Based Offloading.* In this mechanism, the decision to offload is based on the application characteristics, such as the computation intensity, data size, and latency requirements. This can involve partitioning the application into different modules or components, and deciding which components to offload and to where.

(2) *Context-Based Offloading.* In this mechanism, the decision to offload is based on the context of the mobile device, such as the network conditions, battery level, and location. This can involve using sensors and other contextual information to determine when and where to offload.

(3) *In-Network Offloading.* In this mechanism, the decision to offload is made by the network itself, based on various criteria such as network load, traffic congestion, and proximity to computing resources. This can involve using techniques such as network function virtualization (NFV) [48], software-defined networking (SDN) [13].

(4) *Hybrid Offloading.* In this mechanism, a combination of different offloading strategies is used to optimize performance, energy consumption, and other criteria. This can involve dynamically switching between different offloading modes based on the current conditions and requirements.

7.3.4 Edge Serverless Offloading

Now that some underlying concepts about offloading has been discussed, it is time to apply this discussion in a serverless environment. To this extent, offloading of FaaS in edge environments can be quite challenging due to the requirements mentioned. Moreover, there are a number of studies that have already optimized serverless functions to perform offloading for improving performance and lowering energy consumption in edge devices.

NanoLambda [27] proposed a new architecture that enables the deployment of serverless applications on a wide range of IoT devices, from low-power sensors to high-end gateways, by providing an efficient and scalable way of offloading based on workload evaluations in a serverless function.

LETO [36] proposed an approach that uses a combination of serverless computing and energy harvesting to enable efficient task offloading in IoT systems. Specifically, a dynamic optimization algorithm is used to determine the optimal offloading strategy for each computation task, taking into account the energy state of the IoT device, and the network latency.

Baresi et al. [10] proposed a serverless edge computing architecture that leverages offloading to improve the performance of low-latency applications. In this architecture, edge devices are responsible for collecting and processing data in real-time, and for triggering serverless functions to perform additional processing and analysis as needed.

Pelle et al. [55] proposed a dynamic deployment mechanism that uses telemetry data to determine the optimal location for executing serverless functions based on network conditions and device capabilities. The goal is to minimize the overall latency and energy consumption of offloaded tasks, while ensuring high QoS.

Understanding offloading can be a very beneficial component to optimize serverless functions at the edge, as this could lead to lower response time and higher throughput.

8 Application Models

So far, resource management and isolation mechanisms have only discussed as part of the adaption of serverless at the edge. However, it is also important to have a better understanding of the types of applications, the domains and the frameworks that are available in serverless, and find how it could be deployed in smaller devices. As

[11] https://cloudlets.io/

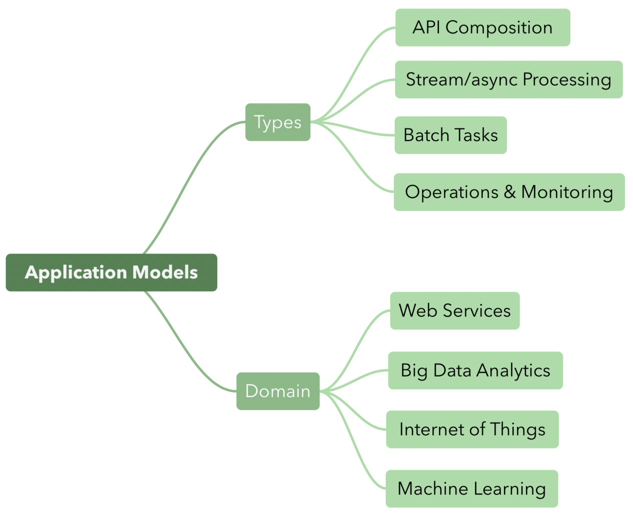

Figure 12: Application Models Taxonomy.

discussed in section 4, **RQ1** breaks down into multiple sub research questions. The third one is as follows:

RQ1.3

What kind of applications models in serverless are usually deployed on edge devices?

This subsection provides an overview of with all the application related concepts that are currently studied in serverless. While doing so, the idea is to find existing optimizations for different application models, with the goal of carrying computation more efficiently in smaller devices. As a result, Figure 12 provides a taxonomy which contains the breakdown of these components.

Eismann et al. [21] analyzed 89 use cases from different data sources with the objective of finding areas within serverless that could be further researched. Hence, their findings would be referenced throughout this subsection in order to have a better picture of the available application models in serverless. It would then be the challenge to see how this could fit in an edge environment. Serverless application models can be generally classified within two different sub-branches: Types and Domains.

8.1 Application Types

There are several distinctive application types available in serverless. They describe the kind of tasks that should be employed by a function when executing incoming workloads. In general, these tasks adhere to the design of serverless functions. For example, jobs with long-running processes would not be best suited on a serverless function, since a function is meant for quick and short executions. Similarly, the objective is to find the types of applications for which serverless performs best and apply them to an edge environment, taking into account the already mentioned requirements at the edge.

8.1.1 API Composition Serverless functions have become very popular for API compositions. Due to its scalable design, serverless functions turn out to be a great option for building web interfaces that experience unpredictable and occasional spikes. FaaS is great for burst workloads, where it can handle multiple concurrent invocations from an endpoint at the same time. This corresponds to 28% Figure 14 of the use cases studied in [21].

Use Case. a mobile application may use several different APIs to pull the weather forecast based on various parameters, e.g. geolocation. Therefore, instead of invoking individual endpoints, which can be resource consuming, a serverless function could be used to aggregate this data, saving energy in the device.

8.1.2 Stream/Async Processing This type is particularly chosen for event driven processes that need to be executed as soon as they arrive in a system. This type of application has enabled event-driven architectures built on serverless functions for certain use cases. Therefore, this is a great use case for edge computing. This corresponds to 27% Figure 14 of the use cases studied in [21].

Use Case. a stream of events generated in a traffic light can be backed by a serverless function. The function would then react based on certain changes in the event, e.g. light changing from red to green, allowing pedestrians to cross the street. Serverless computing can be very useful in such cases where computation is based on changes in events.

8.1.3 Batch Tasks The functionalities from FaaS can be extended to batch processing application types. Certain serverless platforms, such as AWS Lambda, provide parallel execution, through concurrency rate, integrated in their underlying engine. This allows a number of functions to be executed in parallel at the same time, improving performance and reducing processing time. This corresponds to 23% Figure 14 of the use cases studied in [21].

Use Case. a customer analytics platform for a food chain company is trying to get some insights on the purchasing behaviors of their clients. To do, the application needs to analyze millions of data points simultaneously in order to find this behavior. It could then leverage serverless functions to process such batches in a fast and efficient manner.

8.1.4 Operating & Monitoring Serverless functions can also be used to automate many of the tasks associated with operating and monitoring applications and services. Due to their invocation model, it becomes very cost-effective to use functions for checking the status of processes in a system through health or log monitoring. This corresponds to 20% Figure 14 of the use cases studied in [21].

Use Case. a function could be used to periodically check the status of certain component from a system by performing a pinging mechanism. If it captures any problems with the current service, the function could either remediate it, by either invoking another function responsible for such job, or informing the operator.

Overall, serverless functions are leveraged today for a variety of workloads. As the field develops in the following years, there will be an evolution in regard to the types of applications that serverless can support. However, for the specific focus of edge computing, the type of applications mostly used are for **Streaming/Async Processing**.

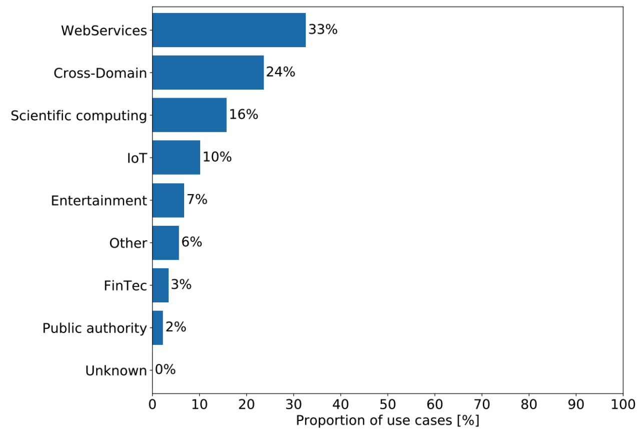

Figure 13: Application Domain Distribution [21].

8.2 Application Domains

Now that the types of applications in serverless have been laid out, it is time to turn the focus towards the different application domains for which serverless and edge can be used. These domains will correspond to a particular type of application, depending on the workload and the use cases. However, the most important discussion is to find how these domains are implemented within both field, serverless and edge. Further, the goal is to arrive at some relationship between both and the specific domain studied.

8.2.1 Web Services

In *Serverless Computing*, Web Services are usually implemented through an API Composition. The implementation of a web service in a serverless function can be done in a variety of programming languages, with JavaScript and Python being the most chosen [22]. Usually these web service functions would be triggered through events, e.g. HTTP or MQTT. They would carry certain business logic for processing workloads, and then return a response to the client. They are generally lightweight, with a relatively short execution time. This corresponds to 33% Figure 13 of the use cases studied in [21].

In *Edge Computing*, Web Services are becoming increasingly common due to the low latency and high throughput that edge computing devices provide. It can be used to offload certain compute tasks from a centralized processing unit, such a cloud, to a local and smaller processing device. For example, edge computing devices could be levered as CDNs to serve static files closer to where the request is originated. This can yield a better performance by providing a lower response time.

8.2.2 Big Data Analytics

In *Serverless Computing*, Big Data Analytics workloads are becoming more popular due to the advancement in performance and scalability from FaaS platforms. As a result, FaaS has grown to support workloads beyond the lightweight use-cases it was originally intended for, and now serves as a viable paradigm for big data processing [37]. **Flint** [34] presented a prototype Spark execution engine that is completely implemented using AWS Lambda and other services. One of the challenges about implementing Spark in serverless is redistribution of data across

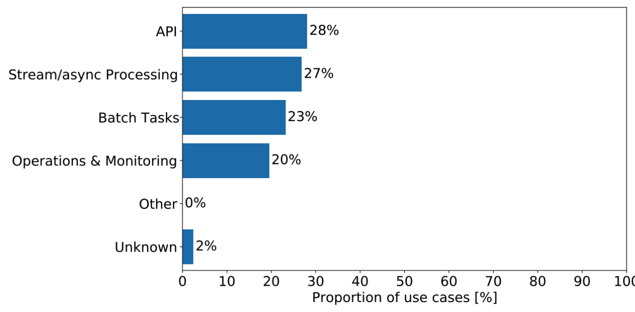

Figure 14: Application Type Distribution [21].

the nodes in a cluster. Since functions are ephemeral [9], it becomes very difficult to perform such operations. However, Flint takes advantage of distributed message queues to handle shuffling of intermediate data.

In *Edge Computing*, Big Data Analytics workloads has been adopted in a smaller scale compared to cloud, due to the resource constraint capabilities already mentioned. However, it does provide the benefit of processing and analyzing data locally where it originated. Edge devices can leverage the compute capabilities when there is no network connectivity to the cloud. **BEGIN** [71] proposed a framework for energy-efficient vehicular edge computing that leverages big data analytics techniques to reduce the energy consumption of vehicles and improve traffic efficiency. The authors propose a system architecture that includes edge computing nodes installed in vehicles and on-road infrastructure, as well as a cloud-based big data analytics platform for processing and analyzing data.

In *Serverless Edge Computing*, Big Data Analytics has become an area of research due to the underlying benefits from both serverless and edge. Nastic et al. [51] proposed a novel approach to implementing a serverless real-time data analytics platform for edge computing. The proposed platform uses a combination of edge devices and cloud-based services to enable real-time processing and analysis of data streams generated by edge devices. The architecture proposed consists of edge devices that collect and preprocess data, a serverless computing platform that provides real-time data processing and analysis, and a data store for storing and retrieving processed data.

8.2.3 Machine Learning

In *Serverless Computing*, this area that has been adopted due to the scalability, cost-effectiveness and reduction of operational complexity that FaaS platforms provide. Carreira et al. [15] summarized the ML workflow consisting of dataset preprocessing, followed by model training and finally by hyperparameter tuning. These workflows have been traditionally deployed in clusters of VMs, however there are drawbacks such as resource management and over provisioning. As a result, serverless provides the benefit of automatic scaling, fine-grained billing, and simpler deployment and management that could be leveraged in ML. **LambdaML** [31] explored the feasibility and performance of serverless architectures for machine learning training. By doing so, through experiments, it was found that serverless architectures

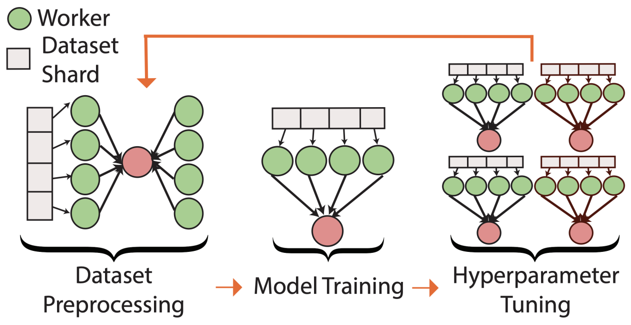

Figure 15: Machine Learning Workflow [15].

can provide reasonable performance for machine learning training tasks, but that there are a number of design choices that can impact the performance of these architectures, such as programming language or training methodology.

In *Edge Computing*, Machine Learning (ML) is becoming a viable option for performing modeling, training and inference in smaller devices. Despite the limitations already mentioned from edge devices, ML workloads can still leverage the low latency and data transferred that are offered in edge computing, achieving real-time data processing performance. Furthermore, decentralization can ML workflows more robust by providing transient services during a network failure [50]. **Rocket** [2] proposed a real-time video analytics system which process live camera feeds from traffic interceptions in Bellevue, Washington. The proposed solution generates directional traffic volumes and raises alerts on anomalous traffic patterns. The goal from this is to identify dangerous conflict patterns in order to minimize accidents and traffic deaths.

In *Serverless Edge Computing*, Machine Learning has already been employed for lightweight processing and analysis of data models deployed in serverless functions at the edge. **STOIC** [70] proposed a framework for executing ML applications in IoT-cloud settings using serverless architecture. The framework leverages an algorithm for scheduling and placement of serverless functions in edge devices and public cloud, specifically focusing on image-based object recognition applications using TensorFlow [12]. When the scheduler at the edge controller receives a batch of images from open field camera traps, it predicts the total response time for processing the batch based on batch size and historical log data. Bac et al. [7] proposed an architecture for deploying machine learning workload as serverless functions in the edge environment. Evaluations were performed to grasp the efficiency of this framework by studying image classification with the MNIST dataset. Results can indicate that the serverless approach can guarantee optimal response and running time, reduce the end-to-end delay of the machine learning application, and show its capability to support distributed machine learning.

8.2.4 Internet of Things

In *Serverless Computing*, Internet of Things is perhaps one of the most notable domains for which serverless functions can be used. IoT devices are continuously capturing data from their environment and surroundings. This data is usually sent to a microprocessor, which acts as a hub for performing computation. Serverless functions can then be invoked through events, very similar to the Web Service domain, to carry certain processing based on the incoming workloads. Furthermore, statelessness of serverless functions add portability for parts of applications to be moved across the edge/cloud computing network with lesser complications [44].

In *Edge Computing*, Internet of Things has become a recent field due to the advancement in technologies in edge computing devices. Some of the characteristics for which IoT benefits from distributed edge are location awareness and low latency. Furthermore, edge computing has the ability to support these requirements of mobility and geographic independence, which are core in a multi sensor deployment system [16].

In *Serverless Edge Computing*, the use cases for which both of these field span is increasing rapidly. Today, there are a lot of deployments for IoT setups computation using serverless function. Due to the benefits of serverless functions, as previously pointed out, there can be a lot of potential for smaller computation and lightweight execution processing in smaller IoT devices. Furthermore, this can present as an opportunity on how to apply energy efficient computations in devices that have resource constraint capabilities. As a result, there are multiple cases to analyze in the field.

Smart Factories are increasingly using connected devices and sensors to collect large amounts of data about their operations. Today, 77%[13] of manufacturers are strongly committed to deploying sensors for their day-to-day operations. As a result, these operations can be used to automate production processes, reduce downtime, and improve quality control. By deploying small, low-power computing devices at the edge of the network, smart factories can perform data processing and analytics tasks at the edge device. Furthermore, serverless computing provides the scalable and elastic computational execution that applications and services may require in smart factories use cases. This can reduce infrastructure costs and provide more flexibility in scaling up or down as needed. Some practical examples in the manufacturing industry can be:

- *Quality Control:* Edge devices can capture visual data from cameras and use computer vision algorithms to detect defects and anomalies in real-time. Serverless functions can be used to analyze this data and trigger alerts or initiate quality control workflows.

- *Inventory Management:* Edge devices can capture data on inventory levels in real-time and use machine learning algorithms to predict future demand. Serverless functions can be used to analyze this data and trigger alerts or initiate inventory management workflows.

- *Supply Chain Optimization:* Edge devices can capture data on supply chain operations in real-time and use machine learning algorithms to optimize logistics and reduce costs. Serverless

[12]https://www.tensorflow.org/

[13]https://www.iiot-world.com/industrial-iot/connected-industry/report-on-state-of-iiot-adoption-and-maturity-in-three-industries/

functions can be used to analyze this data and trigger alerts or initiate supply chain optimization workflows.

Smart Cities are rapidly adopting sensing devices for precise inform-making decisions based on all the data captured. However, managing and analyzing this vast amount of data poses significant challenges, which is where edge and serverless computing can play a key role in enabling more efficient and effective data processing and analysis. Serverless functions at the edge provides the necessary computational power and flexibility to process and analyze data in real-time, which is critical for making timely and informed decisions. By deploying functions on edge devices at various locations throughout a city, data can be collected and processed locally, thus reducing the need to transmit large amounts of data to a centralized location. Some practical examples of smart cities can be:

- *Traffic management:* Edge computing can be used to analyze traffic data from sensors and cameras at intersections to optimize traffic flow and reduce congestion. Serverless computing can be used to process and store this data in real-time.
- *Air quality monitoring:* Edge devices can be used to monitor air quality in different parts of the city, and serverless computing can be used to analyze this data and provide real-time information to residents.
- *Waste management:* Sensors on trash cans and recycling bins can send data to edge devices, which can use serverless computing to optimize waste collection routes and reduce costs.
- *Public safety:* Edge devices can be used to monitor for emergencies such as fires, floods, or even gunshot detection. Serverless computing can be used to quickly analyze this data and alert first responders in real-time.
- *Smart lighting:* Edge devices can be used to monitor ambient light levels and adjust street lighting accordingly. Serverless computing can be used to analyze this data and optimize energy usage.

Smart Farming has been adopting more sensors and IoT devices to make their processes and cultivation of crops a lot more efficient. Farmers are able to make more informed decisions based on the real-time data that these sensors capture. However, similar to the previous cases, scalability is an issue when dealing with a multi deployment sensor IoT application. This can easily cause overload on edge devices if efficient computational mechanisms are not taken into account. Some practical examples of smart farming are:

- *Cop Monitoring:* IoT sensors can be deployed on the farm to collect data on soil moisture, temperature, humidity, and other environmental factors. This data can be processed using serverless computing to provide real-time insights on crop health and growth, allowing farmers to make more informed decisions on irrigation, fertilization, and other crop management practices.
- *Livestock Management:* Cameras can be used to monitor the health and behavior of livestock, collecting data on factors such as body temperature, heart rate, and movement. This data can be analyzed using serverless computing to identify potential health issues, such as disease or injury, allowing farmers to take proactive measures to protect the health of their animals.
- *Predictive Maintenance:* Edge devices can be used to monitor the health and performance of farm equipment, collecting data on factors such as engine temperature, oil pressure, and fuel consumption. This data can be analyzed using serverless computing to predict when equipment is likely to fail, allowing farmers to schedule maintenance proactively and avoid costly downtime.
- *Precision Agriculture:* Edge devices can be used to collect data on soil conditions, crop growth, and weather patterns, which can be analyzed using serverless computing to optimize crop yields and reduce waste. For example, by analyzing data on soil moisture and nutrient levels, farmers can adjust irrigation and fertilization practices to maximize crop growth and reduce the risk of over- or under-watering.

9 Use Case: Precise Agriculture

The main goal of this section is to provide a landscape on how edge computing has done significant technical advances in the agricultural field. The term **Agriculture Technology** (AgTech [14]) is an upcoming field that has been coined to certain agricultural practices that have been already modernized with some sort of edge devices, such as IoT sensors. Therefore, it is crucial to evaluate how serverless computing has fit in such edge environments within the agricultural field. However, before jumping further in the discussion, there are some shortcomings from traditional agriculture that should be understood in order to grasp the full benefits of new technology implementations or integrations.

9.1 Traditional Agriculture Shortcomings

Even though agricultural practices for crop cultivation have changed over the last century, primarily due to the adoption of motor vehicles, it is still a relatively slow moving field where technological innovation happens sporadically. Therefore, this lack of rapid change brings some shortcomings that should be studied in order to highlight the benefits of edge computing adoption for precise measurements.

- *Inefficient use of resources:* Traditional agriculture typically relies on knowledge and approximation from farmers when trying to apply water, fertilizer and pesticides to their crops. This results in an inefficient use of resources, which can certainly lead to environmental damages.
- *Lack of real-time monitoring and control:* Traditional agriculture often relies on manual methods of monitoring and control, which can be time-consuming and inaccurate. This can lead to missed opportunities for early detection and intervention in the case of pest infestations, disease outbreaks, or other issues.

[14]https://www.mckinsey.com/industries/agriculture/our-insights/agtech-breaking-down-the-farmer-adoption-dilemma

- *Limited data collection and analysis:* Traditional agriculture typically involves collecting data from a limited number of sensors or devices, such as weather stations or soil moisture sensors. This can limit the ability to collect and analyze data on a more granular level, such as individual plant health or nutrient levels.
- *Difficulty in identifying and responding to changes:* Traditional agriculture can struggle to respond quickly to changing conditions, such as changes in weather patterns or soil conditions. As a result, this could lead to lack of adjustemnt in crop management practices which can result in lower yields or reduction in crop quality.

These are just a few examples of some of the pitfalls from traditional agriculture. Hence, it is clear that technology, specially edge computing, can bring a lot of benefits to overcome some of these challenges. However, there are certain requirements in the agricultural field that edge computing has to take into consideration.

9.2 Agricultural System Requirements

Agriculture can be a very complicated sector to implement technology, specially due to the limitations in network connectivity and availability. Therefore, it would be important to lay some common requirements that other literatures have addressed in their work, particularly focusing on edge and serverless. The following are just few of the many, depending on the types of systems that need to be built.

(1) *Low latency:* Edge computing devices must be able to process data quickly and efficiently to provide real-time insights for timely decision-making.

(2) *Energy efficiency:* Devices must be designed to consume low power to ensure longevity of the system and reduction in maintenance.

(3) *Connectivity:* The edge devices in a multi-sensor IoT system should be able to communicate with each other seamlessly and with the cloud when needed.

(4) *Data processing:* Edge computing devices should be capable of processing data from different types of sensors, such as weather sensors, soil moisture sensors, and plant growth sensors, and extract meaningful insights from the data.

(5) *Reliability:* Edge computing devices must be reliable and able to operate without interruption, even in harsh environmental conditions.

(6) *Scalability:* Edge computing devices need to adapt tot he number of deployed sensors in the field, without compromising the underlying processing operations or QoS.

These requirements are important when building systems that need to adapt to the ecosystem of a greenhouse or a farm. Henceforth, the main part of the discussion is how edge computing adapt in such cases, and how serverless could be integrated for efficient and low power computation.

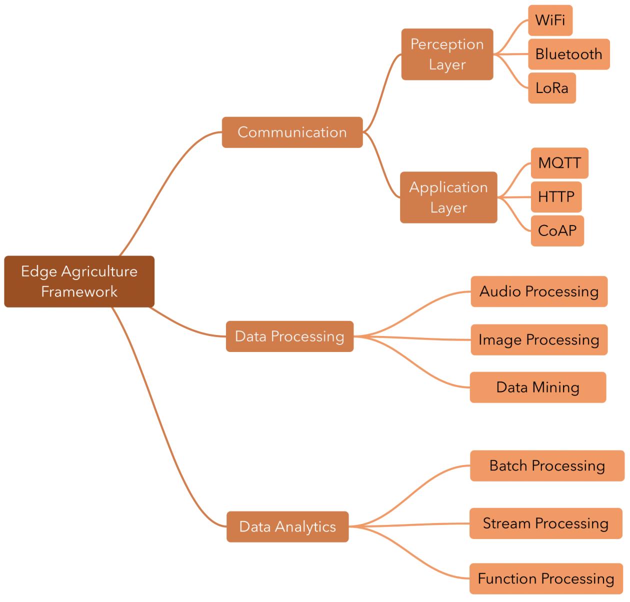

Figure 16: Edge Agricultural Framework Taxonomy.

As part of the main research question, to find the different implementations that edge computing can have in agricultural field, a taxonomy with the most important concepts have been established, which can be seen in figure Figure 16. This depicts three different layers, mainly communications, data processing and analytics. There would be in depth reference in this framework, which would be very useful to understand in order to gauge the capabilities from distributed edge computing in an agricultural domain environment.

9.3 Communication Protocols in Agricultural Systems

Network communication is usually sparse in an open or remote field. Therefore, it is important to find the right network protocols or optimizations that literatures have evaluated when implementing edge computing in such environments. As discussed in section 4, **RQ2** breaks down into multiple sub research questions. The first one is as follows:

RQ2.1

What kind of communication protocols are implemented in edge computing devices when deployed in agriculture fields?

As a result, network communication can be done at two different levels in edge computing, mainly perception and application. The former deals with sensors communication, whereas the latter is more for applications running at the edge. Figure y provides a taxonomy with these two different layers and their corresponding classifications. However, it is important to have a clear distinction among the network protocols available at the edge. There are two different layers to consider, perception and application layers.

Parameter	Wi-Fi	Bluetooth	ZigBee	LoRa
Standard	IEEE 802.11 a, b, g, n	802.15.1	802.15.4	802.15.4g
Frequency	2.4GHz	2.4GHz	868/915 MHz, 2.4 GHz	133/868/915 MHz
Data rate	2–54 Mbps	1–24 Mbps	20–250 kbps	0.3–50 kbps
Transmission Range	20–100 m	8–10 m	10–20 m	>500 m
Topology	Star	Star	Tree, star, mesh	Star
Power Consumption	High	Medium	Low	Very Low
Cost	Low	Low	Low	Low

Table 5: Network communication protocols in IoT for Perception Layer [23]

9.3.1 Perception Layer Protocols

This layer is mostly related to how sensing devices capture data and send it to a centralized hub, such as the cloud. The protocols used in this layer are so-called Wireless Sensing Networks, since they allow wireless communication between sensing nodes and the application layer [52]. They can be further divided in the following categories:

Short-Range Protocols. These protocols usually have a high data transmission and low power consumption [23]. These are used when devices are near each other to communicate. Examples of these can be:

(1) *Wi-Fi.* connects devices in a local area network (LAN) or the internet. It provides high bandwidth and is suitable for devices that require fast data transfer. In agricultural systems, Wi-Fi can be used to connect edge devices to a local network or the cloud, allowing for remote data collection and control.

(2) *Bluetooth.* is a wireless communication protocol designed for short-range communication between devices. Bluetooth's sensors can be deployed in the field to monitor environmental parameters, such as temperature or humidity.

(3) *ZigBee.* is a wireless communication protocol that is designed for low-power, low-data-rate wireless networking applications. In agricultural systems, Zigbee can be used for sensor data collection, or environmental monitoring.

Cellular Networks. Protocols for cellular networks enable communication in long distances with high data transmission rate. However, they have a high power consumption and costs for licensing. Among some examples can be:

(1) *GPRS (2G).* is a wireless communication protocol that is used for data transmission over mobile networks. GPRS can be used to provide wireless connectivity for edge devices that are deployed in remote areas, where other network protocols may not be available.

(2) *3G.* offers higher data transfer rates than 2G technologies, such as GPRS. 3G networks can offer data transfer rates of up to several Mbps, making it suitable for more data-intensive applications. Furthermore, it can be used to provide wireless connectivity to edge devices that require higher bandwidths, such as cameras or drones used for monitoring crops.

(3) *4G/LTE.* is a wireless communication technology that offers higher data transfer rates and lower latency. Furthermore, it can be used to provide high-speed wireless connectivity for edge devices that require real-time data transmission, such as sensors, cameras, and drones used for monitoring crops, weather conditions, and livestock.

(4) *5G.* is the latest wireless communication technology that offers faster data transfer rates, higher capacity, and lower latency than previous generations. With 5G, edge devices used in precise agriculture, such as sensors, drones, and autonomous tractors, can transmit and receive data in real-time at extremely high speeds.

However, it is true that many of the farms and greenhouses today do not have cellular coverage, making this protocol a bit more difficult to adapt to the requirements in the field [46].

Long-Range Protocols. Protocols for long-range networks enable communication in long distances. These protocols can have the lowest power consumption, however data transmission rate is usually pretty low. Examples of these can be:

(1) *LoRaWAN.* is a low-power, long-range wireless communication protocol that is particularly suited for applications that require low data rates and long battery life. It has gained popularity in the agricultural field due to its ability to support long-range communication between edge devices and gateways, making it suitable for monitoring large areas of farmland

(2) *Sigfox.* low-power, wide-area network (LPWAN) protocol that is designed to provide long-range communication with low energy consumption. Sigfox's long-range capabilities make it well-suited for monitoring large areas of farmland, and its low energy consumption makes it a good choice for battery-powered devices that need to operate for long periods of time

9.3.2 Application Layer Protocols

Once data has been captured and sent to a processing unit, other type protocols of protocols need to be evaluated for application communication and presentation purposes. Therefore, the idea would conceptually be the same, finding optimal protocols that can execute much more efficiently in open fields, under the constrained requirements.

(1) *MQTT (Message Queuing Telemetry Transport).* This is a lightweight messaging protocol that is often used in IoT systems to facilitate communication between devices and cloud services. It is designed to be efficient and can work in low-bandwidth or unreliable network conditions.

(2) *CoAP (Constrained Application Protocol).* This is another lightweight protocol that is designed for use in constrained environments, such as low-power devices or networks with limited bandwidth. It is often used in IoT systems to enable communication between edge devices and cloud services.

(3) *HTTP (Hypertext Transfer Protocol).* This is a standard protocol that is widely used on the internet for communication between web servers and clients. It can also be used in agricultural systems to facilitate communication between edge devices and cloud services.

(4) *AMQP (Advanced Message Queuing Protocol).* This is a messaging protocol that is designed to support reliable, asynchronous communication between distributed systems. It is often used in enterprise systems and can be used in agricultural systems to enable communication between edge devices and cloud services.

Table 5 provides a more global comparison of all the communication protocols previously discussed, with their corresponding frequency, data rate, range, power consumption among other factors. Based on this analysis, for short range communication protocols, ZigBee seems to be the most adopted option due to its medium power consumption. On the other hand, LoRa seems to be the most used communication protocol for long range, due to its low power consumption.

9.4 Data Processing

Data processing is another point to consider in this Edge Agricultural Framework. Since there are a variety of sensors that can be used for different purposes, for example image recognition from cows in a farm, it is best to paint a conceptual idea on what type of data measurements can be processed at the edge. As discussed in section 4, **RQ2** breaks down into multiple sub research questions. The second one is as follows:

RQ2.2

What kind of data processing applications can be leveraged in edge computing when deployed in agriculture domain?

These different kinds of processing techniques can make a huge impact in the outcome of the agricultural practices, for instance saving a plant from drought. They can be classified by the following:

9.4.1 Audio Processing Data gathered from microphones can be crucial for studying the behavior of some areas within agriculture. The following are the different use cases for processing recorded data at the edge.

(1) *Pest detection:* Audio signals can be used to detect the presence of pests in crops. For example, the sounds made by corn root worm beetles can be recorded and analyzed to detect their presence in cornfields.

(2) *Disease detection:* Similar to pest detection, audio signals can be used to detect the presence of diseases in crops. For example, the sounds made by rice plants infected with bacterial leaf blight have a distinct pattern that can be identified through audio analysis.

(3) *Livestock monitoring:* Audio signals can be used to monitor the health and well-being of livestock. For example, sensors can be placed in barns or pastures to monitor the sounds made by animals, allowing farmers to detect signs of distress or illness.

(4) *Crop growth monitoring:* Audio sensors can be used to monitor the growth and development of crops. For example, the sounds made by plants as they grow and photosynthesize can be analyzed to provide insights into their health and growth patterns.

9.4.2 Image Processing Image classification is one of the most used techniques for data processing in farms and greenhouses. Images can be very powerful for gathering a wide range of data about crops and livestock. The following are examples that can be considered in precise agriculture.

(1) *Plant health monitoring:* By analyzing images of crops, it is possible to detect signs of disease or stress before they become visible to the naked eye. This can allow farmers to take action before the problem spreads, potentially increasing crop yields.

(2) *Crop counting:* Image analysis can be used to count the number of plants in a given area, allowing farmers to estimate yields and adjust planting density accordingly.

(3) *Weed detection:* By analyzing images of fields, it is possible to detect the presence of weeds and differentiate them from crops. This can allow farmers to target herbicide application more precisely, reducing costs and environmental impact.

(4) *Yield estimation:* By analyzing images of crops throughout the growing season, it is possible to estimate yield potential and predict harvest dates.

(5) *Livestock feed consumption:* Analyzing images of feed bunks or troughs, it is possible to track the amount of feed consumed by the animals and provide automated feeding systems with real-time data to make adjustments as necessary.

9.4.3 Data Mining The data collected from various sensors can be used to identify patterns, correlations, and trends that can provide valuable insights for decision-making in agriculture. Data mining techniques can be applied to extract useful information and knowledge in order to take certain actions, for example.

(1) *Predict crop yields:* By analyzing historical data on weather, soil conditions, and other factors, predictive models can be created to forecast future crop yields. This can help farmers plan for harvests and make decisions on planting and fertilizing.

(2) *Identify disease outbreaks:* Sensors can detect changes in plant or animal behavior that may indicate the presence of disease. Data mining can be used to analyze this information and identify patterns that may indicate an outbreak. Early detection can help prevent the spread of disease and minimize damage to crops or livestock.

(3) *Optimize resource allocation:* By analyzing data on soil moisture, nutrient levels, and other factors, farmers can optimize the use of resources such as water and fertilizer. This can lead to more efficient and sustainable farming practices.

9.5 Data Analytics

Once data has been gathered in different forms, e.g. audio and image, the analysis of this data is a fundamental piece to study. Although this may not specifically apply to the AgTech field, it concerns on the different methods of how data can be analyzed at the edge. As discussed in section 4, **RQ2** breaks down into multiple sub research questions. The third one is as follows:

RQ2.3

What kind of processing techniques can be used at the edge when analyzing data from agriculture field?

There are three main classifications on how data analytics that can be down at the edge. These are summarized in Table 6.

9.5.1 Batch Processing Batch processing engines, such as MapReduce [20], are designed to process bigger datasets efficiently. They use a structured, batched input dataset and perform complex operations on that dataset, for example by distributing computation across multiple nodes. One use case for edge computing and batch processing in agriculture could be to optimize irrigation practices. Sensors in the field can collect data on soil moisture levels, temperature, and humidity, and edge devices can process this data in batches to identify patterns and optimize irrigation schedules. By analyzing the data in batches, insights can be gained on the most efficient times and amounts of water to be applied to the crops, which can help to conserve water and improve crop yields. However, due to the large amount of data required by batch processing jobs, this can be a constraint for smaller computational devices that are bounded by certain battery limitations. Hence, batch processing is not the most optimal choice for preforming data analytics at the edge with a serverless function execution.

9.5.2 Stream Processing Stream processing is a method for real-time data processing, as it is generated by sensors. This is in contrast to batch processing, where data is processed in batches after it has been collected. Stream processing is particularly useful for applications that require immediate or near real-time processing and analysis of data. Some examples of stream processing frameworks that can be used for edge computing in agriculture include Apache Kafka [15], Apache Flink [16], and AWS Kinesis [17]. These frameworks provide the ability to process large volumes of data in real-time, and can be deployed at the edge to enable real-time decision-making. One use case for edge computing and stream processing in agriculture could be in the monitoring and management of livestock. Sensors can be placed on animals to collect data on their health, behavior, and location. Stream processing can then be used to analyze this data in real-time to identify potential health issues in animals. Additionally, stream processing can be used to analyze data on animal behavior and movement patterns to optimize feed for better efficiency and less waste.

9.5.3 Function Processing Function processing directly relates to the serverless computing model discussed in previous sections. Therefore, the idea here is to analyze data from sensors in almost real time. In the context of edge computing in agriculture, function processing can be used to process and analyze data that is generated by sensors in the field or in greenhouses. For example, a function can be triggered by a temperature sensor that detects a sudden drop in temperature, and the function can respond by adjusting the heating system in a greenhouse to maintain optimal growing conditions. Another example of function processing in agriculture is the use of image recognition to identify pests or diseases in crops. An edge device with a camera can capture images of crops, and a function can be triggered to analyze the images and identify any signs of pests or diseases. This can help farmers take corrective action before the problem becomes widespread and causes significant damage to crops. One advantage of function processing in edge computing is that it can reduce latency and improve responsiveness. Since functions can be triggered by specific events or requests, they can be executed quickly and efficiently, without the need for a dedicated server to process data. This can be particularly useful in agriculture, where real-time monitoring and decision-making are critical for maximizing crop yields and minimizing losses.

9.6 Edge Agricultural Systems

Now that requirements and edge agricultural framework have been specified, there is by now a clearer picture on how edge computing could enhance precise agricultural practices. As a result, the primary focus of this section is to build on top of what has already been discussed, in terms of requirements and framework, to provide a complete overview on how edge computing could be adapted to the agricultural field by showcasing three different systems.

9.6.1 FarmBeats: Data-Driven Agriculture [64] . This research projects, developed by Microsoft, aims at helping farmers make more informed decisions by providing them with actionable insights based on real-time data collected from their farms. Figure 17 provides a visual representation of their architecture. The main contribution of the FarmBeats platform is its ability to collect and analyze data from a variety of sources, including IoT sensors, drones, and satellite imagery, and then use machine learning algorithms to provide farmers with insights that can help them optimize their

[15]https://kafka.apache.org/
[16]https://flink.apache.org/
[17]https://aws.amazon.com/kinesis/

	Advantage	Disadvantage
Batch Processing	• Process large volumes of data • Perform operations offline • Can be less resource intensive	• Results not available in real-time • Requires storage large volume of data • Resource constraint on devices
Stream Processing	• Real-time analyzes and process of data • Continuous monitoring of sensors • Detect anomalies or trigger alerts	• Significant processing power • Not suitable to analyze historical data • Susceptible to data loss
Function Processing	• Specific targeted analysis of data • Filtering and transforming data • Lightweight analysis or decision making	• Not suitable for complex analysis • Limited by processing power and memory

Table 6: Data Analysis Technique Comparison

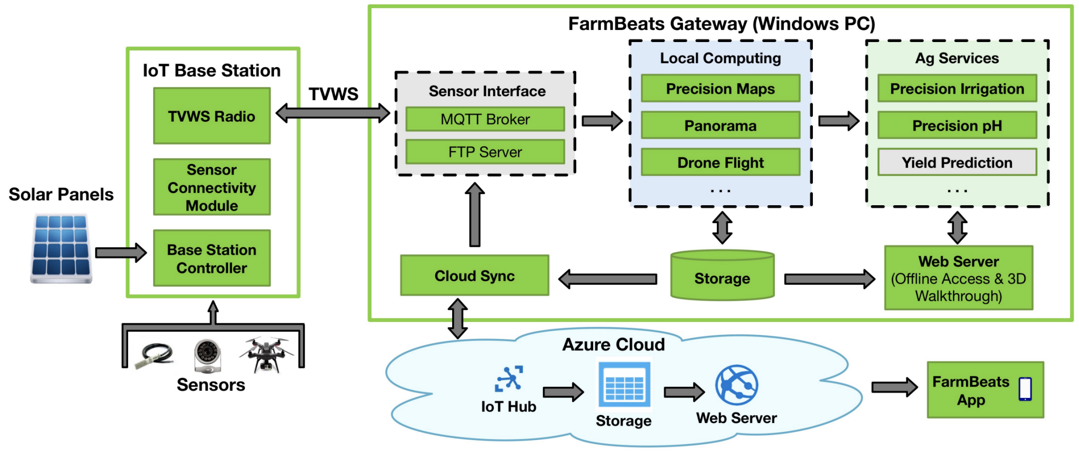

Figure 17: FarmBeats Architecture [64].

operations. There are a variety of key features that can be extracted from this system, which can help to build the discussion on how edge computing works efficiently in remote agricultural fields:

Main Objectives. *Data collection:* the platform is designed to collect data from a variety of sources, including sensors that monitor soil moisture, temperature, and other environmental factors, as well as drones and satellite imagery that provide high-resolution images of crops and fields. *Data analysis:* once the data has been collected, the FarmBeats platform uses machine learning algorithms to analyze it and provide farmers with actionable insights. For example, the platform can predict when a crop is likely to be ready for harvest, or identify areas of a field that are not receiving enough water. *Edge computing:* to ensure that the platform can operate in environments with limited connectivity, FarmBeats uses edge computing to process data locally on the farm, rather than relying on cloud-based services. *Edge computing:* to ensure that the platform can operate in environments with limited connectivity, FarmBeats uses edge computing to process data locally on the farm, rather than relying on cloud-based services. *Integration with existing systems:* the platform is designed to be flexible and can integrate with existing farming systems, including irrigation systems, weather stations, and other IoT devices. *Low Cost:* one of the main objectives of the FarmBeats platform was to reduce the cost of deploying and maintaining IoT sensors on a farm. To achieve this, the platform uses low-cost, off-the-shelf sensors that can be easily installed and maintained by farmers.

Design Decisions. *Scalability:* the platform was designed to be able to handle large amounts of data from a wide variety of sensors and devices, and to be easily scalable as the number of devices and sensors on a farm increases. *Connectivity:* the platform was designed to work in areas with limited or unreliable internet connectivity. To achieve this, FarmBeats uses a combination of technologies, including edge computing, to process data locally on the farm, and LoRaWAN, previously discussed. *Data Security and Privacy:* FarmBeats was designed with data security and privacy in mind. All data collected by the platform is encrypted and stored securely, and farmers have full control over their data. *Interoperability:* the platform was designed to be interoperable with a wide variety of sensors and devices, regardless of the manufacturer or communication protocol used. This allows farmers to easily add new sensors and devices to their farm without worrying about compatibility issues.

Architecture. *Mist layer:* this layer consists of the IoT Base Station, which is responsible for collecting the data from all sensors and devices. In addition, this layer implements **TV White Spaces**, which is a protocol to set up high-bandwidth connection between remote devices [8]. *Edge layer:* this layer primarily includes a gateway that has two purposes: **a)** performs computation locally on the farm data before shipping it to the cloud. **b)** performs independent operation to handle periods of network outage, thus leading to continuous availability for the farmer. *Cloud layer:* this layer includes a set of cloud services that store, process and analyze the data collected from the edge layer. Furthermore, it contains a set of data processing pipelines, machine learning models, and dashboards to provide insights to farmers. *User layer:* this layer consists of the user interfaces and applications that provide farmers with the ability to visualize and interact with the data collected and analyzed by the cloud layer.

9.6.2 Game Theoretic Analysis: Smart Farming [28] .

This research project applied game theory for analyzing the behavior of farms participating in a cooperative smart farming, which would benefit farms to get better insight for enhancing desired output such as crop yielding, water management and irrigation scheduling. Figure 18 provides a visual representation of their architecture. Similar to the FarmBeats research project, there are some common key features that could be extracted from this paper, to grasp a better understanding on how to integrate edge and serverless computing in the agricultural field.

Main Goals. *Sensor network:* the smart farming system should have a sensor network that can collect data from various sources, such as soil moisture, temperature, and humidity. *Decision support system:* the smart farming system should have a decision support system that can provide farmers with recommendations on crop management, fertilization, and irrigation based on the data collected from the sensors. *Communication network:* the smart farming system should have a communication network that can transfer data from the sensor network to the decision support system and to the farmers. *Data security:* the smart farming system should have robust data security measures to protect the privacy and integrity of the data collected from the sensor network. *Cost-effectiveness:* the smart farming system should be cost-effective and affordable for member farms in the cooperative.

Architecture. *IoT Perception layer:* this layer includes the sensors and devices that collect data from the field, such as soil moisture, temperature, and humidity sensors. The data is transmitted to the next layer for processing. *Edge layer:* the layer is composed of a microprocessor Raspberry Pi that enables edge computation as required. This Pi hosts AWS Greengrass deployment, and a customized Lambda function that can run on the gateway. *Cloud layer:* this layer includes the cloud computing platform that processes and analyzes the data collected from the sensors. Data is stored in AWS cloud storage and can be analyzed on individual members of the coop. *Application layer:* this layer includes the decision support system and other applications that provide farmers with recommendations and insights based on the data collected from the field.

9.6.3 Ubiquitous Sensor Network Platform [24] .

This work was focuses on building on scalable platform, able to acquire, process, store and monitor data from crop growing systems using a mobile ubiquitous approach. Implementing precise agriculture requires the collection of large amounts of data, which can be challenging to interpret with many IoT sensors deployed in the field. Thus, the system aims to providing the proper condition for crop development based on atmospheric temperature, luminance, humidity and water PH level. **??** provides a visual representation of their architecture. Just as the previous two systems, there are some key contributions that can be collected, which would help identify the main objectives and design decisions in this agricultural space, specially considering edge and serverless computing.

Design Decisions. *Ubiquitous data collection:* the platform needed to be designed to collect data from multiple locations in a precise agriculture environment. This required the use of multiple wireless sensor nodes that could be distributed throughout the farm. *Scalability:* the platform needed to be scalable to accommodate many sensor nodes as well as different types of sensors. To achieve this, the platform's architecture was designed to be modular, allowing for easy expansion and customization. *Low power consumption:* the sensor nodes needed to be designed to minimize power consumption to prolong battery life and reduce maintenance needs. To achieve this, the sensor nodes were designed to be low power and to transmit data only when necessary. *Robustness:* the sensor nodes needed to be robust enough to withstand harsh outdoor conditions, such as high temperature and humidity, as well as dust and water. The design of the nodes, therefore, included measures to ensure their resilience against environmental factors. *Data security:* as the collected data could be sensitive and confidential, the platform needed to be designed with data security in mind. This included measures such as secure data transmission and storage, access control, and encryption.

Architecture. *Wireless sensor nodes:* these are small devices that are placed throughout a precise agriculture environment to collect data from various sensors. The nodes are designed to be low power and to transmit data wirelessly to a gateway. *Gateway:* this is a device that receives data from the wireless sensor nodes and sends it to the cloud-based server for processing and analysis. The gateway is responsible for aggregating and transmitting data from multiple nodes and can also perform some data preprocessing tasks. *Cloud-based server:* this is the central component of the platform, where the collected data is stored, processed, and analyzed. The server uses various algorithms and models to analyze the data and provide insights into crop growth and health. The server can also send alerts and notifications to farmers or other stakeholders based on the analyzed data. *User interface:* the platform includes a user interface that allows farmers and other stakeholders to view and interact with the collected data. The interface can display real-time data, historical trends, and other metrics, as well as provide tools for decision-making and crop management.

10 Conclusion

With the rise of IoT sensors and edge devices across multiple industries, in particular agriculture, it has become a necessity to

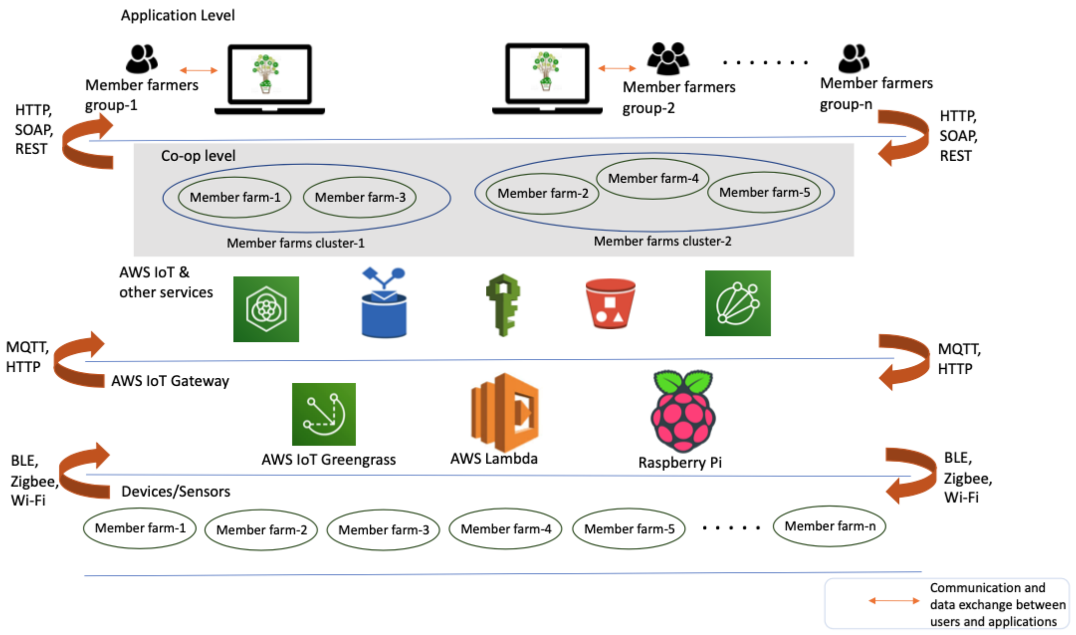

Figure 18: Game Theoretic Analysis for Cooperative Smart Farming Architecture [28].

make computation more efficient and better performing. Therefore, serverless computing is considered an option to achieve these requirements due to its lightweight sandbox environment. Throughout this survey, serverless functions and edge computing have been studied from multiple angles in order to explore the potential scalability and efficiency in smaller devices. As part of this survey, a systematic review has been conducted with the goal of answering two essential research questions: **(1) How can serverless functions be optimized to run at the edge? (2) What sort of implementations are there in agricultural edge computing?**

In the first research question, it was determined that *Unikernels* and *Web Assembly* were better isolation mechanisms, while *Dynamic schedulers* are predominately more optimal for such a changing edge environment. Similarly, *Offloading* mechanisms should be evaluated based on the amount of data that needs to be processed, which could then be from edge to cloud or from cloud to edge.

In the second research question, three main branches were evaluated to understand more about edge computing role in precise agriculture. Communication protocols should take into account efficiency in data transfer. Data processing applications can be used for a variety of purposes, such as audio, image or data mining. Data analytics can be done using batch, stream or function processing.

Future work includes experimenting with different frameworks and optimizations discussed in this survey, in order to find the most efficient serverless function implementation for edge devices. Specifically, the objective is to experiment with edge devices in precise agriculture, where serverless functions would be responsible for data processing and analyzes from sensors.

References

[1] Mania Abdi, Samuel Ginzburg, Charles Lin, José M Faleiro, Íñigo Goiri, Gohar Chaudhry, Ricardo Bianchini, Daniel S Berger, and Rodrigo Fonseca. 2023. Palette Load Balancing: Locality Hints for Serverless Functions. (2023).

[2] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-time video analytics: The killer app for edge computing. *computer* 50, 10 (2017), 58–67.

[3] Mohammad S Aslanpour, Sukhpal Singh Gill, and Adel N Toosi. 2020. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet of Things* 12 (2020), 100273.

[4] Mohammad Sadegh Aslanpour, Adel N Toosi, Muhammad Aamir Cheema, and Raj Gaire. 2022. Energy-aware resource scheduling for serverless edge computing. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 190–199.

[5] Mohammad S Aslanpour, Adel N Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. 2021. Serverless edge computing: vision and challenges. In *2021 Australasian Computer Science Week Multiconference*. 1–10.

[6] Gabriel Aumala, Edwin Boza, Luis Ortiz-Avilés, Gustavo Totoy, and Cristina Abad. 2019. Beyond load balancing: Package-aware scheduling for serverless platforms. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 282–291.

[7] Ta Phuong Bac, Minh Ngoc Tran, and YoungHan Kim. 2022. Serverless computing approach for deploying machine learning applications in edge layer. In *2022 International Conference on Information Networking (ICOIN)*. IEEE, 396–401.

[8] Paramvir Bahl, Ranveer Chandra, Thomas Moscibroda, Rohan Murty, and Matt Welsh. 2009. White space networking with wi-fi like connectivity. *ACM SIGCOMM Computer Communication Review* 39, 4 (2009), 27–38.

[9] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. 2017. Serverless computing: Current trends and open problems. In *Research advances in cloud computing*. Springer, 1–20.

[10] Luciano Baresi, Danilo Filgueira Mendonça, and Martin Garriga. 2017. Empowering low-latency applications through a serverless edge computing architecture. In *Service-Oriented and Cloud Computing: 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017, Proceedings 6*. Springer, 196–210.

[11] Luciano Baresi and Danilo Filgueira Mendonça. 2019. Towards a serverless platform for edge computing. In *2019 IEEE International Conference on Fog Computing (ICFC)*. IEEE, 1–10.

[12] Jeff Barr. 2018. *Firecracker – Lightweight Virtualization for Serverless Computing.* https://aws.amazon.com/blogs/aws/firecracker-lightweight-virtualization-for-serverless-computing/

[13] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. 2016. Software-defined networking (SDN): a survey. *Security and communication networks* 9, 18 (2016), 5803–5833.

[14] David Bermbach, Jonathan Bader, Jonathan Hasenburg, Tobias Pfandzelter, and Lauritz Thamsen. 2022. AuctionWhisk: Using an auction-inspired approach for function placement in serverless fog platforms. *Software: Practice and Experience* 52, 5 (2022), 1143–1169.

[15] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2018. A case for serverless machine learning. In *Workshop on Systems for ML and Open Source Software at NeurIPS*, Vol. 2018. 2–8.

[16] Baotong Chen, Jiafu Wan, Antonio Celesti, Di Li, Haider Abbas, and Qin Zhang. 2018. Edge computing in IoT-based manufacturing. *IEEE Communications Magazine* 56, 9 (2018), 103–109.

[17] Peijin Cong, Junlong Zhou, Liying Li, Kun Cao, Tongquan Wei, and Keqin Li. 2020. A survey of hierarchical energy optimization for mobile edge computing: A perspective from end devices to the cloud. *ACM Computing Surveys (CSUR)* 53, 2 (2020), 1–44.

[18] Jaime Dantas, Hamzeh Khazaei, and Marin Litoiu. 2022. GreenLAC: Resource-Aware Dynamic Load Balancer for Serverless Edge Computing Platforms. (2022).

[19] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2020. Xanadu: Mitigating cascading cold starts in serverless function chain deployments. In *Proceedings of the 21st International Middleware Conference*. 356–370.

[20] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[21] Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. 2020. A review of serverless use cases and their characteristics. *arXiv preprint arXiv:2008.11110* (2020).

[22] Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. 2020. Serverless applications: Why, when, and how? *IEEE Software* 38, 1 (2020), 32–39.

[23] Luis Manuel Fernández-Ahumada, Jose Ramírez-Faz, Marcos Torres-Romero, and Rafael López-Luque. 2019. Proposal for the design of monitoring and operating irrigation networks based on IoT, cloud computing and free hardware technologies. *Sensors* 19, 10 (2019), 2318.

[24] Francisco Javier Ferrández-Pastor, Juan Manuel García-Chamizo, Mario Nieto-Hidalgo, Jerónimo Mora-Pascual, and José Mora-Martínez. 2016. Developing ubiquitous sensor network platform using internet of things: Application in precision agriculture. *Sensors* 16, 7 (2016), 1141.

[25] Philipp Gackstatter, Pantelis A Frangoudis, and Schahram Dustdar. 2022. Pushing serverless to the edge with webassembly runtimes. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 140–149.

[26] Phani Kishore Gadepalli, Sean McBride, Gregor Peach, Ludmila Cherkasova, and Gabriel Parmer. 2020. Sledge: A serverless-first, light-weight wasm runtime for the edge. In *Proceedings of the 21st International Middleware Conference*. 265–279.

[27] Gareth George, Fatih Bakir, Rich Wolski, and Chandra Krintz. 2020. Nanolambda: Implementing functions as a service at all resource scales for the internet of things.. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 220–231.

[28] Deepti Gupta, Paras Bhatt, and Smriti Bhatt. 2020. A game theoretic analysis for cooperative smart farming. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2350–2359.

[29] Chris Hickman. 2020. The Future of Containers – What's Next? https://mobycast.fm/the-future-of-containers-whats-next/

[30] Matthijs Jansen, Auday Al-Dulaimy, Alessandro V Papadopoulos, Animesh Trivedi, and Alexandru Iosup. 2022. The SPEC-RG Reference Architecture for the Edge Continuum. *arXiv preprint arXiv:2207.04159* (2022).

[31] Jiawei Jiang, Shaoduo Gan, Yue Liu, Fanlin Wang, Gustavo Alonso, Ana Klimovic, Ankit Singla, Wentao Wu, and Ce Zhang. 2021. Towards demystifying serverless machine learning training. In *Proceedings of the 2021 International Conference on Management of Data*. 857–871.

[32] Staffs Keele et al. 2007. *Guidelines for performing systematic literature reviews in software engineering.* Technical Report. Technical report, ver. 2.3 ebse technical report. ebse.

[33] Abhishek Khanna and Sanmeet Kaur. 2019. Evolution of Internet of Things (IoT) and its significant impact in the field of Precision Agriculture. *Computers and Electronics in Agriculture* 157 (2019), 218–231. https://doi.org/10.1016/j.compag.2018.12.039

[34] Youngbin Kim and Jimmy Lin. 2018. Serverless data analytics with flint. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 451–455.

[35] Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov. 2014. OSv—optimizing the operating system for virtual machines. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 61–72.

[36] Haneul Ko, Sangheon Pack, and Victor CM Leung. 2021. Performance optimization of serverless computing for latency-guaranteed and energy-efficient task offloading in energy harvesting industrial IoT. *IEEE Internet of Things Journal* (2021).

[37] Jörn Kuhlenkamp, Sebastian Werner, Maria C Borges, Karim El Tal, and Stefan Tai. 2019. An evaluation of faas platforms as a foundation for serverless big data processing. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 1–9.

[38] Nicolas Lacasse. 2018. Open Sourcing Gvisor: A Sandboxed Container Runtime. https://cloud.google.com/blog/products/identity-security/open-sourcing-gvisor-a-sandboxed-container-runtime

[39] Philipp Leitner, Erik Wittern, Josef Spillner, and Waldemar Hummer. 2019. A mixed-method empirical study of Function-as-a-Service software development in industrial practice. *Journal of Systems and Software* 149 (2019), 340–359.

[40] Zijun Li, Linsong Guo, Jiagan Cheng, Quan Chen, BingSheng He, and Minyi Guo. 2022. The serverless computing survey: A technical primer for design architecture. *ACM Computing Surveys (CSUR)* 54, 10s (2022), 1–34.

[41] Wei Ling, Lin Ma, Chen Tian, and Ziang Hu. 2019. Pigeon: A dynamic and efficient serverless and FaaS framework for private cloud. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 1416–1421.

[42] Fang Liu, Guoming Tang, Youhuizi Li, Zhiping Cai, Xingzhou Zhang, and Tongqing Zhou. 2019. A survey on edge computing systems and tools. *Proc. IEEE* 107, 8 (2019), 1537–1562.

[43] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. 2013. Unikernels: Library operating systems for the cloud. *ACM SIGARCH Computer Architecture News* 41, 1 (2013), 461–472.

[44] Anupama Mampage, Shanika Karunasekera, and Rajkumar Buyya. 2022. A holistic view on resource management in serverless computing environments: Taxonomy and future directions. *ACM Computing Surveys (CSUR)* 54, 11s (2022), 1–36.

[45] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. 2017. A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials* 19, 4 (2017), 2322–2358.

[46] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. 2019. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT express* 5, 1 (2019), 1–7.

[47] Philipp Mieden and Philippe Partarrieu. 2019. *Performance analysis of KVM-based microVMs orchestrated by Firecracker and QEMU.* Technical Report. Technical Report. University of Amsterdam.

[48] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. 2015. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials* 18, 1 (2015), 236–262.

[49] Chetankumar Mistry, Bogdan Stelea, Vijay Kumar, and Thomas Pasquier. 2020. Demonstrating the practicality of unikernels to build a serverless platform at the edge. In *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 25–32.

[50] MG Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. 2021. Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–37.

[51] Stefan Nastic, Thomas Rausch, Ognjen Scekic, Schahram Dustdar, Marjan Gusev, Bojana Koteska, Magdalena Kostoska, Boro Jakimovski, Sasko Ristov, and Radu Prodan. 2017. A serverless real-time data analytics platform for edge computing. *IEEE Internet Computing* 21, 4 (2017), 64–71.

[52] Emerson Navarro, Nuno Costa, and António Pereira. 2020. A systematic review of IoT solutions for smart farming. *Sensors* 20, 15 (2020), 4231.

[53] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2018. {SOCK}: Rapid task provisioning with serverless-optimized containers. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 57–70.

[54] Jungae Park, Hyunjune Kim, and Kyungyong Lee. 2020. Evaluating concurrent executions of multiple function-as-a-service runtimes with microvm. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 532–536.

[55] István Pelle, Francesco Paolucci, Balázs Sonkoly, and Filippo Cugini. 2021. Latency-sensitive edge/cloud serverless dynamic deployment over telemetry-based packet-optical network. *IEEE Journal on Selected Areas in Communications* 39, 9 (2021), 2849–2863.

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

[57] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2022. Serverless computing: a survey of opportunities, challenges, and applications. *Comput. Surveys* 54, 11s (2022), 1–32.

[58] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. 2019. Architectural implications of function-as-a-service computing. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*. 1063–1075.

[59] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.

[60] Simon Shillaker and Peter Pietzuch. 2020. Faasm: Lightweight isolation for efficient stateful serverless computing. *arXiv preprint arXiv:2002.09344* (2020).

[61] Christopher Peter Smith, Anshul Jindal, Mohak Chadha, Michael Gerndt, and Shajulin Benedict. 2022. Fado: Faas functions and data orchestrator for multiple serverless edge-cloud clusters. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 17–25.

[62] Amoghvarsha Suresh and Anshul Gandhi. 2019. Fnsched: An efficient scheduler for serverless functions. In *Proceedings of the 5th international workshop on serverless computing*. 19–24.

[63] Bo Tan, Haikun Liu, Jia Rao, Xiaofei Liao, Hai Jin, and Yu Zhang. 2020. Towards lightweight serverless computing via unikernel as a function. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[64] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta N Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. 2017. Farmbeats: An IoT platform for data-driven agriculture.. In *NSDI*, Vol. 17. 515–529.

[65] Laurens Versluis and Alexandru Iosup. 2021. A survey of domains in workflow scheduling in computing infrastructures: Community and keyword analysis, emerging trends, and taxonomies. *Future Generation Computer Systems* 123 (2021), 156–177.

[66] Bin Wang, Ahmed Ali-Eldin, and Prashant Shenoy. 2021. Lass: Running latency sensitive serverless computations at the edge. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*. 239–251.

[67] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 1–10.

[68] Julian Wood. 2020. Building Extensions for AWS Lambda. BuildingExtensionsforAWSLambda Accessed on Feb 4, 2022.

[69] Renchao Xie, Qinqin Tang, Shi Qiao, Han Zhu, F Richard Yu, and Tao Huang. 2021. When serverless computing meets edge computing: architecture, challenges, and open issues. *IEEE Wireless Communications* 28, 5 (2021), 126–133.

[70] Michael Zhang, Chandra Krintz, and Rich Wolski. 2021. Edge-adaptable serverless acceleration for machine learning Internet of Things applications. *Software: Practice and Experience* 51, 9 (2021), 1852–1867.

[71] Zhenyun Zhou, Houjian Yu, Chen Xu, Zheng Chang, Shahid Mumtaz, and Jonathan Rodriguez. 2018. BEGIN: Big data enabled energy-efficient vehicular edge computing. *IEEE Communications Magazine* 56, 12 (2018), 82–89.