

Ecoscape as a Validated Edge-Cloud-Continuum Simulator: An LLM Use-Case Integration and Comparative Study

Darlin Stücker

2025

Kiel Software Engineering Research

KSER, Volume 2

Edited by

Prof. Dr. Wilhelm Hasselbring (hasselbring@email.uni-kiel.de)

DOI: [10.38071/2025-01928-6](https://doi.org/10.38071/2025-01928-6)



This work is published under the Creative Commons Attribution 4.0 International license (<https://creativecommons.org/licenses/by/4.0/deed.en>).

The electronic open access version of this work is permanently available on MACAU (<https://macau.uni-kiel.de>), Open Access Repository of Kiel University

eISSN: 3053-4186

Abstract

The rapid proliferation of Edge Computing demands robust and accurate simulation tools to model and optimize the complex interactions across the Edge-Cloud-Continuum. This thesis addresses the pressing need for a structured evaluation of the current simulation landscape by presenting a comparative study of ten representative state-of-the-art edge computing tools, leading to the deduction of a comprehensive requirements catalogue. This guide establishes a future-proof benchmark for evaluating and developing next-generation edge simulation tools.

Using this newly defined standard, the thesis conducts a targeted validation of Ecoscape. The analysis revealed that Ecoscape satisfies nearly all requirements, demonstrating its high technical maturity, but lacked sufficient representative use-cases to fully showcase its adaptability to emerging technologies.

To address this gap and further validate its utility, this work introduces a novel, timely use-case: A Large Language Model (LLM) workload deployment across the continuum. The integration and subsequent experimental evaluation of this LLM scenario successfully demonstrate Ecoscape's capability to model complex, resource-intensive and modern computing paradigms.

Contents

1	Introduction	1
2	Foundations	3
2.1	Conceptual Foundations	3
2.1.1	Edge Computing Fundamentals	3
2.1.2	Simulation in Computing Systems	8
2.2	Technical Foundations	10
2.2.1	Kubernetes	10
2.2.2	Prometheus and Grafana	11
2.2.3	Apache Kafka	12
2.2.4	Kepler	12
2.2.5	Chaos Mesh	13
3	State-of-the-Art Simulator Analysis	15
3.1	Selection Criteria and Methodology	15
3.2	Representative Simulator Overview	16
3.2.1	EdgeCloudSim	16
3.2.2	iFogSim2	17
3.2.3	YAFS	17
3.2.4	EmuFog	18
3.2.5	Mockfog 2.0	18
3.2.6	Fogify	19
3.2.7	iContinuum	19
3.2.8	FogNetSim++	20
3.2.9	FogTorchII	20
3.2.10	EdgeAISim	21
3.3	Comparative Analysis	21
3.3.1	Analysis Framework	22
3.3.2	Analysis & Results	26
3.3.3	Key Findings & Gap Analysis	41
4	Requirements Catalogue	43
4.1	Architectural Dimensions	43
4.1.1	Mandatory Criteria	43
4.2	Functional Capabilities	44
4.2.1	Mandatory Criteria	44

Contents

4.2.2	Recommended Criteria	44
4.3	Performance and Scalability	45
4.3.1	Mandatory Criteria	45
4.3.2	Recommended Criteria	45
4.4	Usability and User Experience	45
4.4.1	Mandatory Criteria	46
4.4.2	Recommended Criteria	46
4.5	Validation and Evaluation	46
4.5.1	Mandatory Criteria	46
4.5.2	Recommended Criteria	46
5	Use-Case: LLM Deployment	49
5.1	Motivation & Research Question	49
5.2	Design & Architecture	50
5.3	Implementation	51
5.3.1	Challenges	51
5.4	Evaluation Methodology	53
6	Evaluation	55
6.1	Experimental Setup	55
6.1.1	Infrastructure Configuration	55
6.1.2	Evaluation Methodology	56
6.1.3	Workload Generation	56
6.1.4	LLM Service Configuration	56
6.2	Experiments	57
6.2.1	General Observations	57
6.2.2	Scenario 1: Resource Limit Validation	58
6.2.3	Scenario 2: Horizontal Scaling Validation	60
6.2.4	Scenario 3: Heterogeneous Resource Validation	61
6.3	Discussion	63
6.3.1	Limitations	64
7	Conclusion	67
	Bibliography	71

Introduction

Edge computing has emerged as a transformative paradigm that extends computational capabilities from centralized cloud infrastructure to the network's periphery, reducing latency, minimizing bandwidth usage and enabling applications that require immediate response times [Shi et al. 2016]. The development has thereby led edge computing as a complement for cloud computing [Ashouri et al. 2021]. With IoT devices projected to grow exponentially [Shi et al. 2016, Syu et al. 2023], the demand for real-time, low-latency applications, has led organizations to increasingly adopt edge computing architectures to support IoT deployments, autonomous systems, and real-time analytics. Consequently, robust simulation environments have become increasingly critical for system design, performance evaluation, and deployment planning.

The complexity and heterogeneity inherent in edge computing environments present significant challenges for researchers and developers seeking to evaluate system performance, optimize resource allocation, and validate architectural decisions before costly real-world deployments. However, deploying and testing edge computing solutions in real-world scenarios presents substantial obstacles, including high costs, time commitments, and physical constraints that make them highly impractical for research and development purposes [Ashouri et al. 2021, Shi et al. 2016].

Hence, numerous edge computing simulators have been developed within the research community, each offering different foci, distinct capabilities, architectural approaches, and evaluation metrics. These simulators serve as essential tools for analyzing resource allocations, algorithms, configurations, and deployment strategies before committing to physical implementations.

Despite the rapid growth of the edge computing simulation landscape, existing simulators often present different aspects and potentially incomplete or biased simulation environments [Ashouri et al. 2021]. Many simulators focus on specific aspects of edge computing while potentially overlooking other critical factors that influence real-world performance. Additionally, as research progresses in new technologies that become available and standardized like machine learning, containerized applications and orchestrations platforms like for example Kubernetes, the need to extend these technologies to edge environments have become critical. Yet there remains a gap in understanding which aspects are essential or only recommended in the simulation landscape.

The absence of standardized evaluation criteria and comprehensive comparative analyses leaves researchers and developers without clear guidance on which simulators possess

1. Introduction

the necessary characteristics to accurately model edge computing scenarios. This gap in knowledge not only hinders research reproducibility but also potentially leads to suboptimal design decisions based on insufficient simulation foundations.

This is the primary objective of this thesis, in which we establish a framework for evaluating edge computing simulators and guide researchers and developers alike in creating new simulators while additionally demonstrating its practical application through the enhancement of an existing one. This work addresses the fundamental question of what essential accounts for requirements and characteristics that define a comprehensive and effective edge computing simulator.

To answer this question, this thesis begins with the exploration of necessary foundations in Chapter 2. Chapter 3 examines key similarities and differences among current state-of-the-art edge computing simulators across multiple dimensions, providing a comprehensive analysis of the simulation landscape. Through this analysis, Ecoscape is evaluated alongside established simulators to assess its positioning within the current state-of-the-art. Based on patterns identified in the comparison, Chapter 4 presents a requirements catalogue that formalizes the essential characteristics and capabilities of effective edge computing simulators. Chapter 5 addresses a gap identified during the analysis in Ecoscape through development and implementation of a LLM deployment use case that demonstrates the simulator's capabilities. This use-case gets further established by depicting experiments and their evaluation in Chapter 6. Finally, Chapter 7 concludes this thesis with reflections on future work.

Foundations

This chapter establishes the theoretical, in form of conceptual foundations, as well as technical foundations, like the technical components that make up Ecoscape architecture, which are both essential for understanding the presented work in the subsequent chapters.

The first section introduces edge computing fundamentals, including its defining characteristics, architectural models, and the challenges that necessitate simulation-based research approaches as well as a quick glance at a few use cases and applications. The second section explores simulation principles and methodologies, providing the theoretical framework for understanding how edge computing systems can be effectively modeled. Subsequently, the technical foundations section examines the specific technologies employed in Ecoscape.

2.1 Conceptual Foundations

2.1.1 Edge Computing Fundamentals

Definition and Core Characteristics

The European Telecommunications Standards Institute (ETSI) describes edge computing as a “system which provides an IT service environment and cloud-computing capabilities at the edge of an access network [...] in close proximity to its users” [ETSI 2024].

Edge computing therefore represents a distributed computing paradigm that brings computation and data storage closer to the sources of data generation, fundamentally challenging the traditional centralized cloud model. Unlike conventional cloud computing, where processing occurs in distant data centers, edge computing strategically positions computational resources at the network’s periphery (also called the edge), enabling real-time data processing with minimal latency.

The core principle underlying edge computing is data locality, which represents the concept that processing data near its point of origin reduces transmission overhead and improves response times. This approach addresses inherent limitations of cloud-centric architectures, particularly in scenarios requiring immediate decision-making or handling massive data volumes that would overwhelm network bandwidth if transmitted to centralized locations. Edge computing environments present some distinguished key characteristics, such as:

2. Foundations

Ultra-low latency: Edge nodes typically achieve response times of 1-20 milliseconds compared to cloud latencies of 100-500 milliseconds [Li et al. 2017, Ye et al. 2002], making them suitable for time-critical applications such as autonomous vehicle control or industrial automation.

Bandwidth Optimization: By processing data locally, edge computing reduces the volume of data transmitted across wide-area networks, which addresses the growing concern of, for example, network congestion in IoT-dense environments.

Improved Reliability: Due to being locally distributed, typical dependencies like the status of the central cloud server or a stable internet connection are no concerns in edge computing, making services more independent and more reliable.

Enhanced Privacy and Security: Local data processing minimizes exposure of sensitive information during transmission, addressing regulatory compliance requirements and user privacy concerns.

Edge, Fog and Cloud Computing Paradigms

The computing landscape encompasses not only edge computing but also its complementary paradigms, fog and cloud computing, that form together a continuum from centralized to distributed processing.

Cloud computing operates on the principle of centralized resource pooling, where vast computational resources are concentrated in geographically distributed but individually large data center. The model excels in scenarios requiring massive computational power, extensive storage capacity and global accessibility. However, cloud computing faces limitations in latency-sensitive applications, bandwidth-constrained environments and security-sensitive scenarios.

Fog computing on the other hand, is the middle ground between edge and cloud computing, by extending cloud capabilities to the network edge through a hierarchical architecture. Fog nodes serve as intermediate processing layers between end devices and cloud data centers in a region and therefore create an opportunity for regional coordination while maintaining some centralized oversight.

2.1. Conceptual Foundations

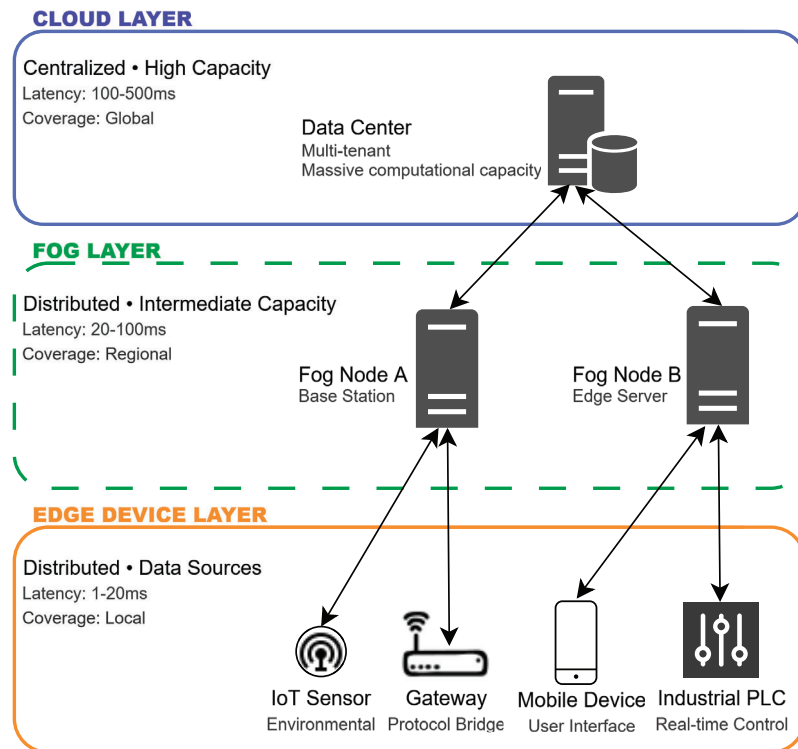


Figure 2.1. Three-tier architecture of cloud, fog and edge computing paradigms illustrating hierarchical processing distribution and data flow patterns.

The distinction between these paradigms often blurs in practice, as modern systems frequently implementing hybrid approaches that leverage the strength of each layer [Shi et al. 2016]. This trend can also be observed in the simulation landscape, presenting a great amount of hybrid simulators.

Architectures and Deployments

Edge computing architectures exhibit significant diversity, reflecting the varied requirements of different application domains and deployment scenarios. However, most implementations share common structural elements that can be categorized into distinct architectural patterns.

Three-Tier Architecture represents the prior described, showcased in 2.1 and most prevalent design pattern, consisting of cloud, fog/edge and device layers. In this model, lightweight edge fog nodes manage regional coordination and data aggregation, while cloud resources provide global oversight and resource-intensive computations. This hierar-

2. Foundations

chical approach balances computational efficiency with management complexity.

Cloudlet-Based Architecture focuses on the small-scale data centers positioned at network access points such as WiFi access points or cellular base stations. These so called cloudlets provide VM-based services to nearby mobile devices, offering a clean abstraction that simplifies application development while maintaining proximity benefits [Satyanarayanan et al. 2009].

Container-Based Edge Deployments have gained prominence due to their lightweight nature and orchestration capabilities. Platforms like Kubernetes have been adapted for edge environments, enabling dynamic service deployment and management across heterogeneous edge infrastructure. This approach particularly suits scenarios requiring frequent application updates or multi-tenant environments.

Serverless Edge Computing represents an emerging pattern where edge resources are abstracted as function-as-a-service platforms. This model simplifies development by allowing developers to focus on business logic rather than infrastructure management, while the platform handles resource allocation and scaling decisions.

Deployment considerations vary significantly based on physical constraints, connectivity patterns and operational requirements. Industrial deployments often emphasize robustness and real-time guarantees, while mobile edge deployments prioritize energy efficiency and seamless handoff capabilities.

Challenges and Requirements

Edge computing introduces unique technical and operational challenges that distinguish it from traditional distributed systems. These challenges stem from the fundamental trade-offs between bringing computation closer to data sources and maintaining system manageability and reliability.

Shi et al. 2016 describes different challenges that need to be addressed when working in this field:

Resource Heterogeneity describes the pool of different resource constraints regarding a wide range of diverse device types in edge environments. Unlike cloud data centers with standardized hardware configurations, edge deployments span diverse device types, from resource-constrained IoT devices to powerful edge servers. This heterogeneity complicates application deployment, performance prediction and resource management strategies.

Network Variability in edge environments exceeds that of traditional distributed systems due to diverse connectivity options ranging from high-speed fiber connections to intermittent cellular or WiFi links. Applications must adapt to varying bandwidth, latency and reliability characteristics often requiring sophisticated fault tolerance mechanisms.

Distributed Management Complexity increases exponentially with the number of edge locations. Traditional centralized management approaches become impractical when managing thousands of geographically distributed edge nodes, necessitating new approaches to monitoring, updating and maintaining edge infrastructure.

2.1. Conceptual Foundations

Security and Trust concerns are amplified in edge environments due to their geographically distribution leading to an increased attack surface and the physical accessibility of edge devices. Edge nodes often operate in unsecured locations requiring robust security mechanisms that function independently of central authority.

Data Consistency and Synchronization challenges arise as the edge is a distributed system which inherits the general problem when multiple nodes must maintain coherent views of shared data. Additionally, due to limited and intermittent connectivity to central coordination services, these challenges are further complicated and can be challenging to tackle.

Use Cases and Applications

Edge computing has found widespread adoption across diverse application domains, each presenting unique requirements that drive the evolution of edge computing architectures and technologies. As different applications impose varying constraints on latency, reliability, processing capacity and data handling characteristics, a quick glance at the supported projects can present a valuable insight and knowledge for further discussions.

Autonomous and Connected Vehicles represent one of the most known and most demanding edge computing applications requiring ultra-low latency decision-making for safety-critical operations. Edge nodes deployed at roadside infrastructure and within vehicles themselves must coordinate to provide seamless connectivity during high-speed mobility scenarios. The computational requirements span from simple sensor data aggregation to complex computer vision and machine learning inference for object detection and path planning as well as quick decision making to guarantee the safety of everyone inside as well as outside of said vehicle [Ibn-Khedher et al. 2021, XIE et al. 2024].

Industrial Internet of Things (IIoT) and Smart Manufacturing applications leverage edge computing to achieve real-time process control and predictive maintenance in manufacturing environments. These deployments often require deterministic response times for safety interlocks and process control loops, demanding robust real-time operating systems and fault-tolerant architectures [Savaglio et al. 2024].

Smart City Infrastructure inherits a broad range of edge computing applications including intelligent traffic management, environmental monitoring and public safety systems. These services extend not only to traffic but also smart waste management, air quality monitoring systems, pollution level observation and many more. The distributed nature of smart city applications and their strong connection to IoT devices requires sophisticated coordination mechanisms to manage city-wide optimization while maintaining local autonomy for critical services [Jararweh et al. 2020].

Augmented and Virtual Reality (AR/VR) applications push the boundaries of edge computing performance requirements while demanding both ultra-low latency and high computational throughput for real-time rendering and content delivery. Mobile AR applications require strong computational resources near the user to compute complex computer vision algorithms and 3D rendering. Multi-user VR environments present additional

2. Foundations

challenges by maintaining synchronized shared virtual spaces across distributed edge infrastructure [Herabad et al. 2024, G.Palanikumar et al. 2025].

Healthcare and Telemedicine applications utilize edge computing to enable real-time patient monitoring and remote medical services while addressing strict data privacy and regulatory compliance requirements. Wearable medical devices and IoT sensors generate continuous streams of personal medical data that must be processed locally to detect any anomalies in patients and trigger immediate alerts when necessary to guarantee the safety of the patients and detect emergencies that may go under the radar when patients are alone. Edge-based medical analysis can provide preliminary diagnoses in remote locations where access to centralized medical expertise is limited, while ensuring that sensitive patient data remains within controlled environments [Prabhu and Hanumanthaiah 2022].

These diverse application domains illustrate a few of the broad spectrum of possible projects and their respective requirements that edge computing system must address. Each use case presents distinct challenges for edge computing simulators, requiring different modeling approaches for network characteristics, computational workloads, mobility patterns and failure scenarios. Understanding the spectrum provides essential context for developing comprehensive evaluation criteria for edge computing simulation tools.

2.1.2 Simulation in Computing Systems

Principles of Discrete Event Simulation

Discrete Event Simulation (DES) forms the theoretical foundation for most edge computing simulators, providing a mathematical framework for modeling complex distributed systems through time-ordered event processing. Unlike continuous simulation models that evolve system state gradually over time, DES advances system state instantaneously at discrete points in time when events occur.

At its core, DES is an event-driven execution model, where system evolution is governed by a chronologically ordered sequence of events. Each event represents a point in time where a system state change occur, such as a task arrival, task process, completion of computation or further network message delivery. This approach particularly suits distributed computing systems, such as edge computing, as state changes are typically triggered by discrete occurrences rather than continuous processes.

For this approach, multiple mechanisms are essential for a functional DES-based simulator[Babulak and Wang 2008, Robinson 2005] :

Event Scheduling mechanisms form the core of any DES system, as the events are the heart of the approach. The event scheduler maintains a priority queue of future events, ordered by their scheduled execution time. When an event triggers, it may modify the system state and schedule additional future events, extending the sequence of related system behaviors. This process terminates by either a termination condition or if the event queue becomes empty.

State Management in DES requires careful consideration of when and how system state

2.1. Conceptual Foundations

change occur. The system needs to maintain a consistent view at each simulated time point, which requires all state modifications to be atomic. This principle becomes particularly important in distributed system simulation, where multiple concurrent events must be carefully coordinated to maintain causal relationships.

Additionally, **Random Number Generation** and **Statistical Modeling** are crucial elements for enabling stochastic behaviors inherent in real distributed systems. Proper handling of randomness requires careful seed management for reproducibility, appropriate statistical distribution selection for modeling system phenomena and sufficient simulation run length to achieve statistical significance in results.

Simulation vs Emulation

For evaluating and testing edge computing tools, multiple approaches should be considered depending on the fundamental design choice that significantly impacts the capabilities, accuracy and performance characteristics of edge computing tools. Understanding the distinction in these approaches is crucial for both developing and selecting appropriate evaluation methodologies.

Simulation creates abstract models that capture the essential behavioral characteristics of target systems without implementing actual system components. Here, scalable and controllable can be defined and typically their behavior gets modelled through statistical approximations and simplified abstractions, enabling rapid evaluation of large-scale scenarios that would be impractical to deploy in reality. However, simulated behavior may not accurately reflect the complexity and unpredictability of real system implementations [McGregor 2003].

Emulation, on the other hand, bridges the gap between simulation and real deployment by combining actual software implementations with virtualized or abstract hardware resources. Here, platforms execute real application code and system software while providing controlled, reproducible execution environments. Hence, emulation typically provides higher accuracy and realistic implementation-specific behaviors and interactions that simulations might overlook, at the cost of greater computational resources and limits to the scale of evaluable scenarios [McGregor 2003].

Simulation and emulation are closely interrelated, which results in **Hybrid** approaches being possible. These have emerged to combine the benefits of both methodologies, by using detailed emulation for critical system components while employing simulation for large-scale or less critical elements [McGregor 2003].

The choice between simulation and emulation depends on evaluation objectives, available resources, and required accuracy levels. Development and research focusing on algorithmic performance might benefit from simulation's scalability, while studies evaluating real system deployments might require emulation's higher fidelity.

Although "simulator" in the strict methodological sense refers to abstract system modeling, the term is often used more broadly in the edge-computing literature to denote any tool that enables controlled and repeatable reproduction of system behavior. Under this

2. Foundations

broader interpretation, simulators may also encompass emulation-based and benchmarking frameworks, since these tools likewise "simulate" edge-cloud conditions in a controlled environment. This can also be observed by the tools themselves, with modern tools blurring the distinction by incorporating aspects of both categories at the same time.

To address this terminological misalignment, the remainder of this thesis distinguishes between dominant emulation-based tools, referred to as emulators, and dominant simulation-based tools, referred to as simulators, while using the term tool as an umbrella designation for both. Tools that feature distinct simulation as well as emulation properties will be further referred as hybrids.

2.2 Technical Foundations

Ecoscope represents a modern approach to edge computing simulation, developed by Reiter et al. 2025 to address the growing need for realistic evaluation of containerized edge deployments. While it operates as a tool for statistical performance simulation, its core methodology is based on high-fidelity emulation building upon Kubernetes orchestration platform. Ecoscope integrates Apache Kafka for distributed data streaming, Kepler for energy consumption monitoring, Prometheus for metric collection, Grafana for visualization, and Chaos Mesh for systematic reliability testing. This creates a comprehensive simulation environment that reflects contemporary edge computing infrastructure while using real workloads in a simulated topology making it a modern hybrid approach for edge computing simulation.

In the following, these technologies are further explored to illustrate their direct influence and which aspects of Ecoscope's architecture they cover.

2.2.1 Kubernetes

Kubernetes¹ (also known as K8s) is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications across distributed computing environments by implementing a master-worker architecture [Hightower et al. 2017]. It provides a robust framework for managing microservices architectures by abstracting underlying infrastructure complexity. Core abstractions include Pods, as the smallest deployable units, Services for network endpoints, and Deployments for lifecycle management.

Kubernetes has evolved from a cloud-native container orchestration platform to a fundamental standardized technology for managing distributed edge computing deployments. Its declarative configuration model and extensive ecosystem make it particularly well-suited for the complex requirements of edge infrastructure management [Carrión 2022].

¹<https://kubernetes.io/>

Edge-specific Kubernetes distributions such as K3s² and MicroK8s³ have emerged to address the resource constraints and operational requirements of edge environments [Koziolok and Eskandani 2023]. These lightweight distributions reduce memory footprint and eliminate dependencies on external databases, making them suitable for deployment on resource-constrained edge hardware while maintaining compatibility with standard Kubernetes APIs.

Furthermore, Kubernetes supports custom resource definitions (CRDs) and operators, which provide mechanisms for encoding edge-specific operational knowledge into the Kubernetes control plane. Edge computing operators can therefore automate complex deployment patterns, manage application lifecycle across heterogeneous hardware, and implement domain-specific optimization strategies while maintaining integration with standard Kubernetes tooling.

Additionally, cluster federation and multi-cluster management capabilities of Kubernetes enable coordinated management of edge deployments across multiple geographic locations.

2.2.2 Prometheus and Grafana

Prometheus⁴ provides metrics collection and monitoring capabilities essential for observability in distributed edge computing environments.

Due to Prometheus's pull-based monitoring architecture, agents to maintain persistent connections to central monitoring systems are not required, and with a time-series database, multiple metrics can be stored efficiently across a heterogeneous distributed infrastructure. Additionally, service discovery integration allows Prometheus to automatically discover and monitor new edge nodes and services as they join or leave the deployment, making it adaptable to changes in a dynamic edge environment where nodes frequently change due to mobility, maintenance, or scaling operations.

This is further supported by Grafana's⁵ visualization capabilities, which transform raw metrics data into actionable insights through customizable dashboards, alerting mechanisms, and data exploration interfaces.

This seamless integration between Prometheus and Grafana creates a complete, adaptable, customizable, and observable monitoring stack that enables developers and researchers to understand system behavior, identify anomalies and bottlenecks, and respond to operational issues in real time.

²<https://k3s.io/>

³<https://microk8s.io/>

⁴<https://prometheus.io/>

⁵<https://grafana.com/>

2. Foundations

2.2.3 Apache Kafka

Apache Kafka's⁶ (hereinafter called Kafka) distributed streaming platform capabilities address critical data management requirements in edge computing environments, particularly for scenarios requiring real-time data processing and reliable message delivery across distributed infrastructure.

The platform's partition-based scaling model allows fine-grained control over data locality and processing distribution, which are crucial capabilities for edge scenarios with bandwidth constraints. With the Kafka Connect ecosystem, extensive integration capabilities are provided via edge data sources and sinks, enabling streamlined data pipeline construction without custom coding. The connector framework supports various protocols, such as industrial protocols, IoT platforms, and database systems, making it widely adaptable in different applications.

Sophisticated real-time analytics are enabled due to Kafka Streams processing capabilities within the Kafka ecosystem, reducing the complexity of processing architectures. Hence, stream processing applications can implement complex event processing, data enrichment, and aggregation logic while maintaining the fault tolerance and scalability characteristics of the underlying Kafka platform.

2.2.4 Kepler

Kepler⁷ (Kubernetes-based Efficient Power Level Exporter) represents a specialized monitoring solution designed to provide granular energy consumption visibility in containerized edge environments. This addresses the growing importance of energy efficiency in edge computing deployments.

The energy consumption is calculated based on the ratio of resources used to total energy. The total energy consumption is collected from Running Average Power Limit (RAPL) zones and other hardware sensors. For systems, the power level can then be determined by dividing the total energy into active and idle based on, for example, CPU usage ratio. On the other hand, for processes, the power level can be calculated by dividing the power used by a given resource based on the ratio of the process and system resource utilization. This can not only happen with CPU usage but also, for example, GPU usage and other resources.

Based on that, Kepler can assign each process, system and container an energy consumption level, which becomes particularly valuable in battery-powered edge deployments where energy efficiency directly impact operational sustainability.

Additionally, Machine Learning-Based Energy Modeling capabilities enable energy consumption prediction based on workload characteristics and historical pattern. These predictive models support proactive energy management strategies and enable simulation of energy consumption under various deployment scenarios.

⁶<https://kafka.apache.org/>

⁷<https://sustainable-computing.io/>

2.2.5 Chaos Mesh

Chaos Mesh⁸ provides chaos engineering powered capabilities specifically designed for Kubernetes environments, enabling systematic reliability testing of edge computing systems under various failure conditions and operational stresses.

Under the motto of "Break Your System Constructively", Chaos Mesh allows for simulated abnormalities that might realistically occur, such as network-level chaos experiments and resource-based chaos testing. Here, multiple variables and failures can be simulated, including but not limited to network partitions, latency variations and bandwidth limitations as well as CPU throttling, memory limitations and disk space restrictions. These experiments enable validation of application resilience to the network unreliability characteristic of edge deployments as well as evaluation of degradation behaviors and additional resource management effectiveness.

⁸<https://chaos-mesh.org/>

State-of-the-Art Simulator Analysis

A comprehensive analysis of the current edge computing simulation landscape is the focus of the following sections. The analysis provides the foundation for developing a systematic evaluation framework for edge computing simulators by establishing clear selection criteria for identifying representative simulators from the extensive body of research and development in this domain. Following a systematic selection process, ten representative edge computing simulators and Ecoscape are examined, each representative offering distinct approaches to modeling edge computing environments and validate Ecoscape. The chapter proceeds with detailed simulator profiles that introduce their core architectures, capabilities, and intended use cases, providing readers with essential background for understanding their respective strengths and limitations. To support the systematic comparison of the ten representative simulators, we establish an analysis framework that defines the aspects and categories in which the comparison should take place. Subsequently, a structured comparative analysis examines these simulators in comparison to their competitors, revealing significant patterns and variations within the simulation landscape. These significant key aspects are highlighted, and missing aspects are covered in a comprehensive gap analysis, establishing the foundation for the requirements catalogue developed in the subsequent chapter.

3.1 Selection Criteria and Methodology

As the edge computing simulation landscape is vast, comparing every existing simulator is neither efficient nor sustainable and does not guarantee that meaningful characteristics are highlighted. Hence, we select a number of representative simulators to represent the current state-of-the-art for the edge computing simulation landscape.

For this, we must determine which criteria a simulator must meet to be chosen, and the number of simulators must be sufficient to provide a comprehensive view of the landscape.

The first and most critical criterion is that every simulator in this list must be published and accessible as an open-source project, so that a meaningful analysis is possible. If the simulator is not accessible, no insight can be gained nor can the analysis be reproduced or reviewed. Additionally, if the simulator is open-source, the approach by which certain features are implemented can be further reviewed and later referenced for one's own implementation.

3. State-of-the-Art Simulator Analysis

Secondly, highly cited or well-established simulators should be present in the list of representatives, as a high citation and reference count demonstrates that the simulators have significant impact in research and development. Furthermore, if a simulator is highly referenced, it can be considered a foundation upon which other researchers and developers have built, and can therefore demonstrate fundamental capabilities that are important for every simulator.

Lastly, we should include simulators that fulfill a meaningful niche or represent a recent trend, as they show the current development direction in the field. They also demonstrate the recent adoption of modern technologies in the simulation landscape and provide valuable insight into current challenges faced in edge computing.

Based on these three criteria, we select ten representatives, which we introduce in more detail in the next section.

As a general foundation for our research and analysis, we conduct a literature review of the original paper for each simulator as well as any related papers. This ensures that we have a general theoretical knowledge base for each simulator, as well as their related components, such as simulators they build upon or reference. Additionally, we examine the official implementation and documentation to gain broader insight into each feature and the architecture of each simulator. By testing each simulator, we also ensure that each one is still deployable and works as intended, even if it may be older than others.

3.2 Representative Simulator Overview

In the following, each of the ten representatives gets quickly introduced, by stating which criteria they fulfill and by a quick rundown of their capabilities, foci and current development status as well as their strengths and weaknesses. This section is only to quickly introduce the representatives and a further more detailed rundown is present in the subsequent section as each important capability gets compared.

3.2.1 EdgeCloudSim

EdgeCloudSim [Sonmez et al. 2018] is a highly cited and widely adapted edge computing simulator, which focuses on Mobile Edge Computing (MEC) via cloudlets and general edge computing simulation. With it being first published August 2018, it is one of the older simulators in this list. As of this thesis, EdgeCloudSim was last updated October 2020.

EdgeCloudSim is based on CloudSim [Goyal et al. 2012] and presents a modular architecture with advanced mobility modeling in Java. The general design philosophy is to create a CloudSim-based framework designed for Edge Computing scenarios with a focus on both computational and networking resources. It supports cloud-edge mobile multi-tier topologies and different network types such as WLAN, WAN, and cellular networks.

EdgeCloudSim is specifically designed for MEC scenarios and demonstrates excellent mobility modeling capabilities. Additionally, the mobility modeling is supported by realistic

3.2. Representative Simulator Overview

network modeling for WLAN and WAN with distance-based latency, and due to its modular architecture, EdgeCloudSim allows easy customization. However, development halted as of October 2020, and due to its focus on MEC scenarios, the remaining scope is relatively limited. It is therefore less suitable for pure IoT or cloud computing scenarios and shows limited built-in support for modern artificial intelligence and machine learning workloads. Furthermore, it is only single-threaded, which leads to scalability limitations, and has no security, privacy, or energy modeling.

3.2.2 iFogSim2

iFogSim2 [Mahmud et al. 2021] is another highly cited and widely adapted edge computing simulator, which builds upon its previous version iFogSim [Gupta et al. 2016]. It is, just like EdgeCloudSim, based on CloudSim [Goyal et al. 2012] and focuses also on MEC scenarios but extends its scope to IoT, fog computing and microservices. iFogSim2 is developed as a CloudSim-based framework designed for comprehensive Edge/Fog computing scenarios with focus on mobility, clustering and microservice orchestration, and was first published in September 2021. The development seems to be inactive, as the last meaningful update was done August 2021.

Just as its competitor EdgeCloudSim, it shows an advanced mobility support but provides ways to include real datasets for more realistic approaches. As for topology, iFogSim2 allows for multi-tier cloud-fog-edge-devices and custom topologies with their respective supported network and protocol types.

iFogSim2 presents comprehensive fog/edge computing simulation capabilities and advanced mobility support with it being able to integrate real datasets. It has strong academic backing and is best used in IoT and fog computing research as well as academic prototyping and clustering algorithm development. But iFogSim2 also has its weaknesses: The scalability is limited by its single-threaded execution and it shows limited real-time simulation capabilities. Furthermore, the output is console-only and needs manual implementation for file output. It has a limited built-in security modeling and energy modeling capabilities and no native containerization support. Therefore, it performs relatively poorly for real-time system simulation requirements, massive parallel processing scenarios, pure cloud computing simulations or scenarios requiring extensive energy modeling.

3.2.3 YAFS

Yet Another Fog Simulator (YAFS) [Lera et al. 2019] follows EdgeCloudSim and iFogSim2 as respected foundational simulators in the simulation landscape. First published in July 2019 and last updated June 2022, YAFS is written in Python 2.7 and extends EdgeSimPy [Souza et al. 2023] and focuses on IoT, fog and edge computing as well as resource allocation. YAFS is designed to be lightweight, robust and highly configurable based on complex network theory with minimal class structure.

3. State-of-the-Art Simulator Analysis

It exceeds in being extremely lightweight and being highly configurable and extensible. YAFS shows full transparency of simulation data and allows dynamic control of all simulations aspects. This makes it perfect for research requiring custom algorithms and policies as well as complex network topology analysis and scenarios requiring high customization.

Nonetheless, as YAFS has a relatively inactive development status and is based on a deprecated python version it has some modern flaws. For example, it has no built-in energy modeling nor security or privacy features. Just like its competitors, it struggles with being single-threaded and therefore limits its scalability. With this, YAFS is not ideal for production or commercial environments, large-scale parallel simulations or research that focus on energy or security.

3.2.4 EmuFog

EmuFog [Mayer et al. 2017] is highly cited, widely adopted and well established extensible emulation framework for large-scale fog computing written in Kotlin and based on MaxiNet¹ [Wette et al. 2014] and Mininet² [Lantz et al. 2010]. First published in October 2017 and updated until September 2020, EmuFog focuses on large-scale fog computing infrastructure emulation while mainly addressing the network emulation.

It provides highly accurate results and supports docker containerization and both synthetic and real-world topologies. EmuFog has a good scalable architecture and efficient fog node placement algorithms with cost optimizations, making it a sufficient tool for network performance analysis in fog environments, application deployment and orchestration testing as well as large-scale fog infrastructure evaluation.

On the other hand, EmuFog has no mobility nor energy modeling and requires high computational resources for large-scale emulation. Due to that, large projects are expensive and mobile or energy projects or research is not recommended with EmuFog. Additionally, its overhead is quite high, making it inefficient for quick prototyping and algorithm development.

3.2.5 Mockfog 2.0

Mockfog 2.0 [Hasenburg et al. 2023] is a widely adopted and well established tool in the simulation landscape and is a successor of the previous Mockfog (1.0) [Hasenburg et al. 2019]. It is not a simulator like the previous mentioned, but a cloud-based emulator written in Node.js, which focuses on fog and edge computing for real application testing. The emulator was first published 2021 and the development halted around June 2021.

Mockfog 2.0 presents a real infrastructure emulation in cloud environments to test fog applications under realistic conditions. It features docker containerization and automated experiment orchestration while maintaining a real cloud-based emulation with runtime network manipulation.

¹<https://maxinet.github.io/>

²<https://mininet.org/>

3.2. Representative Simulator Overview

Due to being an emulation, the resulting behavior is realistic and accurate. Additionally, it allows for dynamic, realistic and scalable testing scenarios and reduces manual effort. With this it succeeds in its main goal of real application testing, but also is useful for validation of prototypes, algorithms and performance. Additionally, it can be used for infrastructure capacity planning and industrial research, making it quite adaptable.

Nevertheless, due to having a cloud infrastructure usage it has high cost and a complex setup. It is not suitable for large-scale theoretical studies and cannot test theoretical algorithms easily. Additionally, it has limited mobility support making it inefficient for some projects.

3.2.6 Fogify

Fogify [Symeonides et al. 2020] is a Python based emulator that focuses on fog/edge computing in a cloud-native container orchestration based network emulation. It is designed to be realistic, reproducible and to server scalable fog/edge testbeds using containers. First published in November 2020, Fogify shows a relatively active development with a modern docker and kubernetes based approach for linux networking.

Due to being realistic and reproducible fog/edge testbeds, it shines at testing real applications in such scenarios. Furthermore, due to its container and kubernetes integration it supports declarative scenario description and therefore being relatively easy to configure and quite modern.

It features no built-in energy modeling which makes it insufficient for energy-focused research unless the energy-consumption is instrumented externally. Large-scale and time-accelerated scenarios also struggle with Fogify, as its hardware-limited scalability stands in the way.

3.2.7 iContinuum

iContinuum [Akbari et al. 2024] is an emulation toolkit from 2024 and is designed for the Edge-to-Cloud Continuum that utilizes a hybrid methodology for experimentation. While it achieved high-fidelity replication by leveraging containerization to run actual applications, its core strength lies in its use of Software-Defined Networking (SDN) which allows emulated environments in a strategically modelled simulation. Here, it is also designed to be realistic and reproducible, but other than the previous, focuses on cloud-to-edge, orchestration and reproducibility of large-scale experimentations.

iContinuum succeeds in its goals by serving realistic and reproducible cloud-to-edge continuum testbeds while maintaining a multi-domain orchestration and declarative scenario description. It is therefore suitable for orchestration research across the cloud-edge continuum, testing real applications in distributed scenarios and in network-based resource emulation.

However, just like its competitor, it is hardware-limited in its scalability and has no synthetic time-acceleration like simulators in this field. Additionally, it has no built-in

3. State-of-the-Art Simulator Analysis

energy modeling, making it not as suitable for large-scale, time-accelerated or energy-focused simulation.

3.2.8 FogNetSim++

FogNetSim++ [Qayyum et al. 2018] is relatively well known and established tool written in C++ and first published October 2018. It is based on the OMNeT++³ and INET⁴ Framework and focuses on network-centric fog environments with distributed fog systems. The development halted as of the same month.

FogNetSim++ succeeds in advance network modeling with packet-level precision and with the built on proven OMNeT++ framework it shows extensive networking capabilities. Furthermore, it present excellent scalabilities for large-scale network simulations and broad selection of debugging and visualization tools. Additionally, it also present deterministic and reproducible simulation results. This supports network performance analysis and protocol development as well as large-scale network simulation with fog nodes and academic research that requires high network simulation accuracy.

However, due to being highly reliant on OMNeT++ and INET, further development and customization of the simulator require expertise and shows a steep learning curve and complex setup. Additionally, the documentation is relatively limited and due to the focus primarily laying on networking other components are being neglected. This makes the simulator relatively inefficient for quick prototyping and real-time system simulation as well as energy- and security-focused studies.

3.2.9 FogTorchII

FogTorchII [Buyya and Srirama 2019] is a well established probabilistic analysis tool in graph structure application deployment. Written in Java in May 2017, FogTorchII focuses on fog computing, Quality of Service (QoS) aware deployment optimization as well as IoT application deployment. With its probabilistic QoS-assurance and resource consumption estimation for eligible deployments of fog application with focus on deployment decision support, it serves a unique focus in the simulation landscape. No visible development can be observed since January 2019.

It succeeds in clear separation between infrastructure and application modeling as well as cost-aware deployment optimization. Therefore, it is best used for fog application deployment planning, QoS-aware deployment optimization, cost analysis for fog deployments and general academic research on deployment strategies in the fog computing field.

However, it features no runtime simulation capabilities and the deployment analysis is rather static with no dynamic reconfiguration or adaptation. Furthermore, no energy or security modeling is present and its limited scalability makes it inefficient for energy or security based large-scaled projects.

³<https://omnetpp.org/>

⁴<https://inet.omnetpp.org/>

3.2.10 EdgeAISim

EdgeAISim [Nandhakumar et al. 2024] is a relatively new simulator, first published in October 2023, and focuses on the growing trend of artificial intelligence on the edge. It features an extension to EdgeSimPy [Souza et al. 2023], which is written in Python, and focuses on resource management and energy optimization for artificial intelligence and machine learning workflows on the edge.

The integration of advanced AI models for intelligent resource management as well as the use of advanced artificial intelligence techniques like multi-armed bandit, Deep Q-Networks (DQN) and GNNs, makes it an efficient simulator in regards to scenarios of energy optimization studies, intelligent resource management and task scheduling algorithm development on the edge.

As EdgeAISim is a relatively small extension and has a very strict focus, it lacks in multiple other features that its competitors present. Due to this, scenarios without artificial intelligence requirements don't reap any benefits from this simulator, as the workflow either hindered by additional workload or is simply delegated to EdgeSimPy.

Tool	Language	Type	Focus
EdgeCloudSim	Java	Sim	Mobile Edge Computing, General Edge Computing
iFogSim2	Java	Sim	IoT, Mobile Edge Computing, Microservices
YAFS	Python	Sim	IoT, resource allocation, complex networks theory
EmuFog	Kotlin	Emu	Large-scale fog infrastructure
Mockfog 2.0	Node.js	Emu	Real application testing, Cloud-based emulation
Fogify	Python	Emu	Container orchestration, Network emulation
iContinuum	Python	Hybrid	Orchestration, Large-scale experimentation
FogNetSim++	C++	Sim	Realistic network-centric distributed systems
FogTorchII	Java	Sim	IoT application deployment, QoS-aware deployment optimization
EdgeAISim	Python	Sim	AI/ML Edge Computing, Resource management, Energy optimization
Ecoscope	Python	Hybrid	Fault Tolerance, Remediation Benchmarking

Note: Sim = Simulator, Emu = Emulator

Figure 3.1. Summary of representatives of the edge computing simulation landscape

3.3 Comparative Analysis

In the following, we further dive into each aspect of the representatives and find similarities and differences. To do that, we first introduce our analysis framework, which covers which

3. State-of-the-Art Simulator Analysis

aspects we observe, what value they bring to the simulation landscape and what the general structure for our comparison is. After that we start with the comparison, covering each aspect and dive into further insights that come up during the process. Additionally, we highlight some common patterns and findings as well as mention some gaps due to missing capabilities, limitations and emerging requirements.

3.3.1 Analysis Framework

To systematically compare each representative with different architectures, foci and implementation approaches, we establish a foundation in form of a framework. Given that our selected representatives range from simulators to emulators with varying specialized purposes, direct feature-to-feature comparison would be insufficient. Instead, our framework considers both the specific goals of each representative and the influence of it in the edge computing simulation landscape. A feature of a fog-edge emulator, may not be a great fit for a cloud-edge simulator, but still can have some insights and useful information regarding implementation or general design.

The framework encompasses five core dimensions that collectively address the complete software development life cycle (SDLC). This holistic approach ensures our analysis captures not only what they can do, but also how well it serves researchers and users across diverse edge computing scenarios.

Architectural Dimensions

The architectural foundation determines the flexibility, maintainability and ability to accurately represent edge computing environments. It concerns itself with the design choices during the development of the simulator. This can include the choice of the coverage scope, simulation paradigm, the general design philosophy, architecture patterns and integrations capabilities.

Which layer the simulator covers, is important to understand its purpose and capabilities. The coverage scope not only determines requirements for protocols and other features, but also different use cases and therefore the general focus. It further can determine required modeling complexity, affects scalability requirements and shapes varied approaches during development and usage. Furthermore, some capabilities, requirements and approaches can be essential for a layer in general.

The choice of the simulation paradigm, on the other hand, determines whether simulators employ discrete-event simulation for network dynamics, continuous-time modeling for resource consumption, agent-based approaches for autonomous behavior or hybrid combinations that leverage multiple paradigms simultaneously. As all these different paradigms, can have influence over simulation accuracy, time and resource usage, the selection should be not made hastily.

Hence, the general design philosophy is a critical point, as it can not only determine the simulation paradigm but also can directly influence the focus of the simulator. As we

established, different foci doesn't mean that no insight can be gained for their tool. This is also true for the design philosophy.

Additionally, the choice of a modular or monolithic architecture pattern, is a foundational choice, as modular designs are known to be more maintainable by separating concerns into independent components, and monolithic implementations are deemed to be compacter by integrating all functionality within unified frameworks.

Functional Capabilities

Functional capabilities encompass the breadth and depth of the edge computing scenarios each simulator can represent. This does include general functionalities as well as focus-based capabilities, which needs to be weighted.

The general functionalities encompasses for example network modeling, device and infrastructure modeling and application and workload modeling as well as fault tolerance and reliability modeling and security and privacy concerns. Each of these functionalities can vary in implementation and approach based on multiple factors, just like if simulation or emulation is presented.

Network modeling assesses support for supported topologies, diverse network types, protocol implementations and realistic network conditions including variable delays, bandwidth constraint and congestion-aware behavior. Additionally, simulated topologies and other feature can have impact in the network modeling as well, making it a core point in the simulator of choice.

Device and infrastructure modeling examines the range of supported device types from resource-constrained sensors to powerful edge servers, including resource modeling, heterogeneity support for diverse device capabilities and realistic scaling behavior that reflects hardware constraints and performance characteristics. The degree, in which these devices need to be modeled, is in correlation to the level of detail that the simulator wants to accomplished and its general focus.

Application and workload modeling investigates support for different application architectures, task dependency patterns, data generation and data flow models. This includes evaluation of service migration capabilities, workload generation methods and QoS requirement modeling for latency, throughput and reliability constraints.

This gets extended by modeling fault tolerance and checking for reliability of the simulated devices and networks. If the network modeling concerns itself with the aspects of variable delays, fault tolerance concerns itself with the case of failing nodes and devices that would maybe cause these delays. By modeling fault tolerance, the simulator can check if the scenario is reliable in cases where devices disconnect.

Furthermore, security and privacy of the simulator as well as the application are important for personal use as well as developing security-aware applications. These points can be further extended by focus-based capabilities, as every type of focus can have qualifing capabilities for the edge computing landscape. For example, if the focus lies on mobile devices, mobility modeling is necessary to model different delay and different data

3. State-of-the-Art Simulator Analysis

generation based on movement and distance to other nodes. Energy consumption and sustainability can also be capabilities important for the simulator that need to be observed and weighted based on their respective influence in the edge computing simulation landscape.

Performance and Scalability

Performance characteristics directly impact practical utility for large-scale edge computing studies. A simulator that can't be scaled or performs badly in realistic deployment requirements, can be deemed insufficient. As scalability is a core aspect of the edge, it is a critical aspect that needs to be analysed for each simulation.

Computational efficiency measures simulation execution speed relative to scenario complexity, memory usage patterns and effectiveness of optimization features including caching, checkpointing and incremental simulation capabilities.

Scalability limits determine practical boundaries for scenario complexity, including maximum supportable network nodes, concurrent applications and realistic scenario sizes before performance degrades unacceptably. This assessment examines parallelization support through multi-threading, distributed processing or GPU acceleration and evaluates how performance scales with increasing scenario complexity.

Usability and User Experience

Usability factors significantly influence simulator adoption and research productivity.

An evaluation of the setup and deployment progress assesses installation complexity, dependency management and documentation quality as well as community support availability. This includes examination of platform compatibility, deployment options and the overall barrier to entry for new users.

Programming interface analysis compares configuration methods from graphical interfaces to APIs, evaluating the balance between ease of use and flexibility. The assessment considers API design quality, example scenario availability and the steepness of the learning curve required for productive use across different users expertise levels. This does not only include the pure usage of said simulator but also the further development. Furthermore, it also observes the strictness of the simulator, as a simulator which can only be used for one specific scenario may not be suited in general.

Research workflow integration examines support for experimental design including scenario management, parameter sweep capabilities, reproducibility mechanisms through seed control and deterministic execution and statistical analysis features. Effective tools should provide data export capabilities, visualization support and integration with external analysis tools to support rigorous experimental methodologies and interpretation of the results in general.

Extensibility determines a simulator's ability to evolve with changing research needs and support novel edge computing concepts. This ensures that the simulator grows with

3.3. Comparative Analysis

the changing simulation landscape, adapt modern approaches and can be customized for a variety of scenarios and use cases. Hence, this aspect is highly dependent on other factors such as the architecture pattern but can also be achieved and influenced by other internal as well as external features.

Plugin architecture is a common way of adding new functionality without modifying core code. To assess this architecture, the availability of well-defined extension points, plugin management systems and the ease of integrating custom components need to be observed.

Programming extensibility evaluates API completeness and choice of programming language for custom development, scripting support for automation and the flexibility of extending existing models or creating entirely new behavioral representations. This includes assessment of inheritance mechanisms, interface design and the complexity required for implementing custom device types, protocols or application models.

Data and metric extensibility investigates capabilities for custom metric definitions, support for novel data patterns, integration with real-world datasets and mechanisms for extending output analysis capabilities. Robust extensibility should enable researchers to adapt tools for emerging edge computing paradigms without requiring fundamental architectural changes.

Validation and Evaluation

Validation capabilities ensure simulation credibility and enable meaningful research conclusion. The accuracy of the tools' results is important, as inaccurate results defeat the whole purpose of the process as a whole.

Accuracy and validation assessment examines built-in verification methods, calibration requirements against real-world data and documented accuracy claims with associated error bounds or precision levels.

Benchmarking and comparison evaluates availability of standard benchmark scenarios, reference implementations for comparative analysis and built-in metrics that enable fair evaluation across different approaches. The assessment considers whether tools provide validated baseline scenarios and support for reproducing published research results for comparison, review and transparency of the results.

Metrics and analysis support examines the comprehensiveness of built-in measurement capabilities, configurability of output generation and support of custom metric definition. Effective evaluation support should, as mentioned in previous aspects, include statistical analysis features, data export capabilities in standard formats and visualization tools that facilitate both real-time monitoring and post processing analysis of simulation results.

3. State-of-the-Art Simulator Analysis

3.3.2 Analysis & Results

Having established the analysis framework prior, the following presents a systematic comparison of the selected representative tools while also stating Ecoscape coverage of each aspect.

Architectural Dimensions Comparison

Starting with the architectural dimensions, the representatives demonstrate some architectural diversity and some unity in their approach to edge computing simulation.

Tool	Coverage Scope	Sim. Paradigm	Arch. Pattern
EdgeCloudSim	Cloud-Edge-Mobile	DES	Modular
iFogSim2	Cloud-Fog-Edge-Device	DES	Modular
YAFS	Cloud-Fog-Edge	DES	Modular
EmuFog	Fog-Edge	Emulation	Modular
Mockfog 2.0	Cloud-Fog-Edge	Emulation	Modular
Fogify	Fog-Edge	Emulation	Modular
iContinuum	Cloud-Edge	Emulation	Modular
FogNetSim++	Fog-Edge	DES	Modular
FogTorchII	Cloud-Fog-Edge	DES	Modular
EdgeAISim	Cloud-Fog-Edge	DES	Modular
Ecoscape	Cloud-Edge	DES	Modular

Figure 3.2. Comparison overview for architectural dimensions

As we can see in figure 3.2, all of the selected representatives demonstrate a relatively clear unity in the choice of architectural pattern, as well as the choice of simulation paradigm if they are a simulator and not an emulator.

Every simulator and emulator chose the modular architectural pattern, which allows for easier updates, parallel development and better risk management. Additionally to that, simulators chose the described standard for simulation paradigms Discrete-Event Simulation (DES) as their respective simulation paradigm. With the combination of these two, identification of bottlenecks, updating components and optimize resource allocation as well as improve performance and workflow can be easily done.

However, the differences occur in the coverage scope. As all of the tools cover the edge, none is a pure edge concerned tool but rather use a multi-tier approach with at least two layers, if not more. Especially, the full-stack of cloud, fog and edge is mostly presented, with the combination of fog and edge alone coming second with a relatively good presented foundation. The only outlier currently present is iContinuum with a scope of cloud and edge, leaving fog out.

As the borders between each layer blurs in reality, covering multiple layer does not only

3.3. Comparative Analysis

make the simulator / emulator more realistic but also supports developers and researchers alike, as not multiple different simulator and emulators need to be intertwined to achieve the goal. However, the choice of how many layers should be covered, should be considered and studied based on the focus of the simulator / emulator.

Based on the comparison, at least fog and edge should be covered to present not only the devices on the edge but also connectivity. The ideal would be the full-stack of cloud, fog and edge to have a complete and modern simulator / emulator that covers all potential applications and projects.

Ecoscape in that regard fulfills the observed patterns by being strictly written in modular python modules and allowing discrete event simulation through scheduled arrivals of data served by custom load generators, fault injection via chaos engineering to simulate real-world faults with the help of Chaos Mesh and the measurement of monitored and logged key performance indicators over a simulated duration. Additionally, it covers a multi-tier approach by allowing the definition of resource constraint nodes, making it possible to deploy more restricted edge and less restricted cloud nodes.

3. State-of-the-Art Simulator Analysis

Functional Capabilities Comparison

Network modeling capabilities demonstrate significant variation based on each representative architectural approach and target scenario.

Tool	Network Types	Supported Protocols	Topology Support
EdgeCloudSim	WLAN, WAN, Cellular	HTTP, Custom	Hierarchical
iFogSim2	WiFi, Ethernet, Cellular, Custom	TCP/IP, HTTP, MQTT, Custom	Hierarchical, Custom
YAFS	Generic/Custom	Generic/Custom	Any
EmuFog	WiFi, Ethernet	TCP/IP, HTTP, Custom	Custom, Real-world datasets
Mockfog 2.0	Any	Any	Hierarchical, Custom
Fogify	Ethernet, Wifi, Cellular	TCP, HTTP, MQTT	Hierarchical, Mesh, Custom
iContinuum	Ethernet, Wifi, Cellular	TCP, HTTP, MQTT	Hierarchical, Mesh
FogNetSim++	WiFi, Ethernet, Cellular, WAN, LAN, Custom	TCP/IP, UDP, HTTP, Custom	Hierarchical, Mesh, Tree, Custom
FogTorchII	Abstract	Abstract	Hierarchical, Custom
EdgeAISim	Ethernet, WiFi, Cellular	TCP/IP, HTTP, MQTT	Hierarchical, Custom
Ecoscope	Ethernet, Wifi, LAN, Custom	TCP/IP, HTTP, UDP, Custom	Hierarchical, Custom

Figure 3.3. Comparison overview of network modeling types

Topologies are often depicted as hierarchical with the most presented addition of custom topologies. EdgeCloudSim only supports hierarchical topologies and is therefore the most restrictive simulator in that comparison. Every other simulator and emulator specifically supports either hierarchical or custom topologies, which includes hierarchical. In general, most of the tools include a way to customize and configure topologies, which leads to customizable scenarios and simulations. YAFS represents this aspect the most, as its focus on complex network theory allows it to freely deploy any topology. EmuFog, on the other hand, stands out for its integration of custom as well as real-world datasets as topologies. It uses the topology generator BRITE [Medina et al. 2001] for custom topologies and real-world datasets from CAIDA's Resource Catalog⁵, which makes it highly customizable and realistic as an emulator.

As a general rule, all tools support standardized protocols for their respective network

⁵<https://www.caida.org/>

3.3. Comparative Analysis

types. EdgeCloudSim simplifies network modeling to focus on mobile edge specifics, rather than additional abstraction of network and protocol types. This is also the case in some regards for EdgeAISim and emulators like EmuFog, Fogify, and iContinuum. As the latter are emulators, their respective emulated network is limited. The only exception is Mockfog 2.0, as it is highly configurable and supports any network type and protocols, with the limitation that the deployed network type and protocols must be supported by the underlying cloud infrastructure. EdgeAISim is limited in its network capabilities by EdgeSimPy, which, as its base, does not provide extensive protocols or types. iFogSim2, on the other hand, provides more comprehensive network modeling, as the default supported types are broader and custom network types and protocol types can be achieved.

FogTorchII is an outlier in this regard, as it does not have direct network modeling but rather generic network links with configurable QoS profiles. The same profiles represent the protocols, making FogTorchII not directly comparable to its competitors as a whole.

Tool	Bandwidth Modeling	Delay Modeling
EdgeCloudSim	Congestion-aware, configurable per link	Distance-based, dynamic
iFogSim2	Variable, configurable per link/n-ode	Propagation, queueing, processing delays
YAFS	Configurable, supports dynamic changes	Configurable, supports dynamic changes
EmuFog	Realistic	Realistic
Mockfog 2.0	Realistic	Realistic
Fogify	Realistic	Realistic or configurable
iContinuum	Configurable per link	Configurable per link
FogNetSim++	Configurable, packet-level	Configurable, packet-level
FogTorchII	Probabilistic	Probabilistic
EdgeAISim	Configurable per link	Configurable per link
Ecoscape	Configurable	Configurable

Figure 3.4. Comparison overview of bandwidth and delay modeling approaches

Characteristics of network modeling show broader unity than the network modeling itself. Almost all simulators provide configurable bandwidth modeling per link or node.

Emulators generally depict a realistic network through emulation, while Mockfog 2.0 has further means for configuration. EdgeCloudSim has congestion-aware bandwidth modeling and, based on its focus on MEC, distance-based and dynamic delay modeling. With this combination, it presents a robust and realistic approach for network modeling. iFogSim2 allows for different delay models to be used. It supports propagation, queueing, and processing delays, therefore presenting a greater selection for different scenarios. FogTorchII, on the other hand, provides probabilistic means for bandwidth and delay modeling. It is thereby less granular in detail, as it may not capture fine-grained, event-level

3. State-of-the-Art Simulator Analysis

interactions like packet collisions and protocol-specific behaviors. This makes it limited in realism for protocol and network studies. Additionally, it is rather static and validation may be challenging, as results depend on well-calibrated real-world data. However, probabilistic modeling allows representation of real-world variability if calibrated correctly and may capture uncertainty in network conditions such as fluctuating bandwidth, random delays, and failures.

Device and infrastructure modeling reveals fundamental differences in abstraction levels and resource representations. Emulation platforms generally model real hardware constraints through container resource limits, providing authentic resource contention but with limited abstraction capabilities. Traditional simulators, like EdgeCloudSim, iFogSim2, and FogNetSim++, show detailed and heterogeneous device modeling with CPU, memory, storage, and energy consumption parameters. While specialized tools, like FogTorchII, focus on ML-specific resource characteristics, including GPU and inference accelerators. EdgeAISim is a combination of both aspects, as its base EdgeSimPy adapts traditional simulators and the AI/ML-based focus of EdgeAISim has some regard for GPU and inference accelerators. Lastly, YAFS emphasizes computational flexibility through abstract resource containers, making it highly configurable and abstract.

Thus, with the general problem of heterogeneity on the edge, edge computing tools should provide meaningful ways to model different aspects of devices and their infrastructure to simulate authentic behavior on the edge.

Chaos Mesh and Kubernetes itself cover these for Ecoscape, as the network structure is depending on the Kubernetes deployment and the network behaviour like delays and bandwidth can be tempered with via chaos mesh which allows for a general configurable network. Additionally, due to their nature the network can be updated dynamically depending on the scenario if further influence is needed.

Application and workload patterns also present similar level of variation as the device and infrastructure modeling.

3.3. Comparative Analysis

Tool	Application Architecture	Task Dependencies
EdgeCloudSim	Mobile application patterns, Back-and Foreground task classification	Parent-child relationships
iFogSim2	Monolithic, Microservice	DAG, Precedence constraints
YAFS	Flexible computational graphs	Configurable graph-based
EmuFog	Real Containerized	Docker container linking
MockFog 2.0	Container Orchestration, Real microservices	Container orchestration
Fogify	Kubernetes, Pod-based	Kubernetes service meshes
iContinuum	Abstract Services, Real containerized applications	Cross-service communication
FogNetSim++	Fog-specific	DAG
FogTorchII	ML inference containers	ML pipeline chains
EdgeAISim	Neural networks, Federated learning patterns	AI pipeline dependencies
Ecoscape	Containerized Microservices	Asynchronous Data Flow (Kafka)

Figure 3.5. Comparison overview for application architecture and task dependency

Emulation platforms handle real application architectures through emulation. While EmuFog, Mockfog 2.0 and Fogify built upon their respective containerized approaches with Docker and Kubernetes for real containerized applications, iContinuum brides simulation and emulation with service-oriented application modeling that can represent both abstract services and real containerized application across the computing continuum. Based on the approach, the task dependencies are handled either via the containerize technology. Thus, emulators provide reproducible and robust ways of application and task dependencies by building upon standardized and proofed robust modern technologies.

Traditional simulators on the other hand vary heavily in their approaches. iFogSim2 and FotNetSim+++ build upon Directed-Acyclic Graph (DAG) for Task dependencies, but have different application architectures. FogTorchII and EdgeAISim have a similar relationship, as both present application architecture suited for ML-tasks with task dependencies relying on pipelines in their respective fields.

Traditional simulators succeed in abstract application modeling flexibility and differ more in this domain as emulators that provide authentic application execution. Specialized tools, like EdgeAISim and FogTorchII provide domain-optimized application support with respective pipeline chains and dependencies, while other simulators built upon their respective target and focus group and use established procedures like Directed-acyclic graphs (DAG) and Parent-Child relationships.

Ecoscape fits right in with these tools, as its allowing for for containerized microservices and by allowing asynchronous data flow via kafka to dictate task scheduling and dependencies. Making the process streamlined and robust as well as configurable which

3. State-of-the-Art Simulator Analysis

can also be seen in data generation:

Tool	Data Generation	Data Flow Models
EdgeCloudSim	Synthetic	Mobile data patterns, Location-based
iFogSim2	Synthetic, Integration	Periodic, Bursty, IoT streams
YAFS	Synthetic, Integration	Stream processing, Batch jobs
EmuFog	Integration	Real data processing
MockFog 2.0	Integration	Real-world data integration
Fogify	Integration	Persistent volumes, Config maps
iContinuum	Integration	Real-time data routing
FogNetSim++	Synthetic	Data aggregation, Filtering
FogTorchII	Limited Synthetic	Model I/O, Batch processing
EdgeAISim	Synthetic, Integration	Training data, Inference patterns
Ecoscape	Synthetic, Integration	Custom

Figure 3.6. Comparison overview for data generation and flow models

Data generation is provided either synthetically or with real-world dataset integration. Emulation in general features emulated real-world data, thereby only offering its integration. Traditional simulators, on the other hand, always provide a way for synthetic data generation. EdgeCloudSim has specifically designed workload and Poisson distribution for data generation, while FogNetSim++ focuses on synthetic traffic via OMNeT++ and scenario configuration. Both of these simulators provide no integration of real-world datasets. iFogSim2, YAFS, and EdgeAISim, on the other hand, provide means to generate data synthetically and to integrate real-world datasets. Here, iFogSim2 allows XML configuration and trace-based workloads, YAFS allows for customizable distributions, timestamp arrays, and support of real traces, while EdgeAISim supports AI-driven workload generation and integration of real traces via EdgeSimPy. An outlier here is FogTorchII again, as it focuses more on deployment and is rather an analysis tool; its data generation is very limited for probabilistic deployment analysis and supports only limited runtime data generation.

The data flow is modeled based on the focus of each project. While emulators target real data processing and persistency, simulators target stream processing and inconsistencies in the flow with mobile data patterns and bursty or periodic behavior. Both kinds of flows represent authentic and realistic data behavior, as real-world data is also inconsistent.

This unity in the edge computing simulation landscape can also be witnessed in Ecoscape, as due to its configurable Kubernetes deployment allows for custom load generator that can be synthetically or integrate real-world datasets while also customize the data flow to any extend. This makes Ecoscape highly customizable and adaptable to many applications needs and makes it highly versatile.

Fault Tolerance and reliability support are demonstrated by varying sophistication levels across all evaluated tools.

3.3. Comparative Analysis

Tool	Failure Types	Detection Methods	Recovery Strategies
EdgeCloudSim	Mobile device, Network failure	Timeout	Task migration
iFogSim2	Device, Network failure	Heartbeat, Timeout	Basic restart
YAFS	Configurable failures	Custom detection	Configurable recovery
EmuFog	Container failure	Docker health checks	Container restart
MockFog 2.0	Container/Service failure	Orchestration monitoring	Auto-restart, Isolation
Fogify	Pod/Service failure	Kubernetes probes, Monitoring	Kubernetes healing, Migration
iContinuum	Service failure	Cross-tier monitoring	Service migration
FogNetSim++	Hierarchical failure	Fog-aware monitoring	Redundancy, Failover
FogTorchII	Model deployment failure	Health checks	Model redeployment
EdgeAISim	Model degradation, Byzantine	AI-aware detection	Model recovery, Consensus
Ecoscape	Custom	Custom	Custom

Figure 3.7. Comparison overview for fault tolerance and reliability support

As figure 3.7 presents, emulators mainly focus on container failures that gets detected by monitoring the system and health checks and get recovered by simply restarting the resource. Traditional simulators on the other hand show a more device focused approach by checking the state of the device and application via timeouts, heartbeats and health checks or by monitoring the general network for anomalies. Recovery is often achieved with task migration to other devices or simple restarts of the original failing devices or applications.

Just as data generation, fault tolerance shows a highly customizable structure in Ecoscape. Abstracted by Chaos Mesh, availability for custom failure types is guaranteed while detection methods and recovery strategies are mostly handled by Kubernetes itself. Often these result in simple timeouts and health checks as well as simple pod restarts but can further be modified to fit custom scenarios and requirements.

Security and privacy on the other hand is an aspect that varies drastically across the simulation landscape. Not only in implementation approaches and covered scenarios but also general existence in some tools.

3. State-of-the-Art Simulator Analysis

Tool	Security Mechanisms	Privacy Support	Attack Modeling
EdgeCloudSim	Secure offloading, Transmission encryption	None	None
iFogSim2	Authentication, Encryption overhead, Trust models	Basic data anonymization	Limited, custom extensions possible
YAFS	None, custom extension required	None	None, custom extension required
EmuFog	None, relies on Docker/Mininet	None	None
MockFog 2.0	None	None	None
Fogify	None	None	None
iContinuum	None	None	None
FogNetSim++	None, relies on OM-NeT++/INET	None	None
FogTorchII	None	None	None
EdgeAISim	None, can simulate AI-based anomaly detection	None	None
Ecoscape	None	None	None

Figure 3.8. Comparison overview of default security and privacy measures

Most tools do not present default security mechanisms, privacy support, or attack modeling, and abstract these concerns mostly to the application. The emulators in particular provide no implementation for testing security or privacy in containers and rely heavily on the implemented security measures in their respective technologies. They depict attack modeling only if it is implemented in the application itself. The traditional simulators provide, with the exception of EdgeCloudSim and iFogSim2, no further implementation of security or privacy measures or attack modeling. This is also the case for Ecoscape. YAFS and FogNetSim++, due to their highly customizable nature, show extension points for security and privacy concerns but at the current state do not provide any in the default implementation. EdgeAISim also provides no default implementation of security and privacy measures, but provides an AI-based anomaly detection simulation, as a way to attempt attack detection.

On the other hand, EdgeCloudSim and iFogSim2 provide relatively basic implementations of security mechanisms, that provide more means to simulate cost and overhead of encryptions and no real protocol-level security, which makes their implemented security relatively limited and mostly for research prototyping.

Mobility support is a relatively focus-based capability for tools to have. If the focus

3.3. Comparative Analysis

does not lie on mobile devices, for example in a static deployment, mobility may not be deemed necessary; on the other hand, multiple projects in the field of edge computing and most notably in the field of IoT devices, show usage of mobile devices and therefore their respective simulators provide necessary means to simulate the network and data characteristics of mobility.

Tool	Mobility Mod-els	Handover Sup- port	Location Ser- vices	Dynamic Topology
EdgeCloudSim	Advance cellu- lar models	Advanced	Location- based services	User mobility
iFogSim2	Basic mobility models	Limited	Basic location- awareness	Node mobility
YAFS	Limited	None	None	Dynamic place- ment
EmuFog	None	None	None	None
MockFog 2.0	None	None	None	None
Fogify	None	None	None	None
iContinuum	None	None	None	None
FogNetSim++	Limited	None	None	Limited
FogTorchII	None	None	None	None
EdgeAISim	Limited	None	None	Limited
Ecoscape	None	None	None	None

Figure 3.9. Comparison overview of mobility modeling

EdgeCloudSim and iFogSim2 provide, with their focus on mobile and IoT devices, advanced mobility support for simulations on the edge. Due to their robustness with handovers, location services, and dynamic topologies, they provide essential features for mobility-focused projects. YAFS, EdgeAISim, and FogNetSim++ provide limited or user-extendable mobility, but are not as advanced by default as their previously mentioned competitors. EdgeAISim especially provides no further implementation but relies mainly on its base EdgeSimPy in this regard. However, every other tool, including Ecoscape, disregards mobility entirely and therefore provides no implementation for such scenarios.

Energy consumption, as well, is a relatively sporadic deployed feature in the edge computing simulation landscape.

3. State-of-the-Art Simulator Analysis

Tool	Energy Modeling Scope	Optimization Support	Granularity
EdgeCloudSim	None	None	None
iFogSim2	Basic device-level energy modeling	Energy-aware placement	Per device, per task, limited accuracy
YAFS	None	None	None
EmuFog	None	None	System, per container
MockFog 2.0	None	None	System, per container
Fogify	None	None	System, per container
iContinuum	None	None	System, per container
FogNetSim++	None	None	None
FogTorchII	None	None	None
EdgeAISim	AI/ML workload energy modeling	Energy optimization	Per task, per device
Ecoscope	Custom	Energy-aware scheduling and optimization	Node, Pod / Container, Process

Figure 3.10. Comparison over of energy consumption modeling

Figure 3.10 shows that only iFogSim2 and EdgeAISim provide meaningful implementations for simulating energy consumption for energy-focused scenarios. However, the emulators do not need to simulate energy consumption, as they emulate a real device and therefore can measure real energy usage externally. This provides energy consumption based on total system consumption but also per container or pod and can be extended to per task if instrumented in the application.

Ecoscope abstracts its energy consumption to Kepler which exports real-time granular power metrics to Prometheus.

Contrary to the belief that if we cover mobile devices, we always look out for battery life and energy consumption, EdgeCloudSim does not provide a meaningful energy consumption model. It can only be approximated by resource utilization based on scenario results, but no direct energy model is implemented. However, it is mentioned in the documentation of EdgeCloudSim⁶ as a needed feature for the simulator.

Performance and Scalability Comparison

Performance and scalability are playing a crucial part in software engineering and especially in edge computing simulation, because of the sheer amount of heterogeneous devices which each generates or compute data on the edge. While comprehensive performance benchmarking across edge computing simulators remain limited, each tools respective paper made some effort to show general performance metrics and some comparisons were

⁶<https://github.com/CagataySonmez/EdgeCloudSim>

3.3. Comparative Analysis

executed by other papers [Fahimullah et al. 2023, Mechalikh et al. 2025, Svorobej et al. 2019].

Computational efficiency is a varying factor across the simulator landscape based on architectural design choices, modeling complexity and implementation optimization strategies. YAFS demonstrate superior computational efficiency through its custom-built discrete event engine with optimized data structures and parallel processing capabilities. Traditional simulators that rely on either CloudSim-based frameworks (EdgeCloudSim and iFogSim2), rely heavily on their deploy technologies like FogTorchII and FogNetSim++ or the emulators, are dependent on the performance of their foundations. The CloudSim-based frameworks show moderate performance limited by the framework's single-threaded architecture and extensive object creation in Java, while FogTorchII relies heavily on its containers optimizations and the emulators face inherent efficiency constraints due to real application execution overhead, with Mockfog 2.0 showing the highest resource consumption due to full system emulation. Furthermore, EdgeAISim suffers additionally from the size of AI models, which results in high memory usage but its performance can vary based on deployed model and scenario.

Additionally to note is, that iFogSim2 actively claims to have a 28% better memory usage than its competitors [Mahmud et al. 2021].

Scalability limits, on the other hand, reveal a critical aspect in the edge computing simulation landscape, as its highly impacted by scenarios, simulation properties and hardware constraints. The maximum number of supported nodes is often not clearly stated [Qi et al. 2025], but sometimes implied or in a very few comparison depicted for selected scenarios.

Emulators are typically stated to have a low amount of maximum nodes, as they need to emulate a whole real system rather than simulate only certain aspects. However, emulators may achieve greater number of nodes by distribution [Qi et al. 2025, Hasenburg et al. 2023]. This is especially stated for Mockfog 2.0, as "MockFog's scalability will usually be limited by the number of VMs that can be provisioned from the cloud provider of choice" [Hasenburg et al. 2023]. Otherwise, they mainly rely with their scalability on their underlying technologies and programming languages. This is the case for iContinuum [Akbari et al. 2024]. Fogify and Emufog have both scalability claims, but don't further classify or depict a benchmark for reviewing. Symeonides et al. 2020 states for Fogify that "the emulation execution environment should be scalable from topologies with a limited number of nodes, capable to run on a single laptop or PC, to hundreds or thousands nodes, running on a whole cluster." but don't provide any further necessary means or valueable benchmark to confirm or deny this claim, while Mayer et al. 2017 states something similar with EmuFog being capable to emulate large network topologies, but doesn't further clarify what this assessment mean nor provide any means to review this claim.

For traditional simulators, some papers show comparison based on a shared scenario and simulation settings or just depict general metrics for their respective simulator. The general consensus is, that iFogSim2 and FogNetSim++ can present up to thousands of

3. State-of-the-Art Simulator Analysis

nodes on a large-scale basis, with the correct simulation setting. However, FogNetSim++ will quickly perform worse if the simulation is not correctly fine-tuned, as they simulate a greater network and focus more computational power for this network simulation. EdgeCloudSim is relatively near this large-scale assessment but falls off because of the same reasons, that the focus on mobility has a higher priority than scalability, making it fall off. Hence, in more complex scenarios EdgeCloudSim shows a relatively bad scalability but in the right circumstances large-scale scenarios it presents up to thousands of nodes. FogTorchII and EdgeAISim are mainly limited by their respective workload for AI/ML scenarios and cant be clearly classified.

The most powerful simulator in this list is YAFS, as shown by Lera et al. 2019 in comparison to iFogSim, with large-scale possible scenarios with thousands of nodes. Every source, however, puts scalability in high regards and challenge in edge computing simulation.

Usability and User Experience Comparison

The first step in using a tool is its setup and deployment process. CloudSim-based simulators like iFogSim2 and EdgeCloudSim offer straightforward setup and deployment through standardized Java installation and inclusion of all necessary dependencies. Their setup documentation is relatively useful and they ship with multiple example scenarios that can be easily deployed to check if the setup was successful.

However, it is notable that iFogSim2 recommends IDEs for Windows usage and EdgeCloudSim's documentation is based on some videos rather than written text. FogNetSim++ is relatively complex, especially on Windows operating systems, with its key dependencies based on OMNeT++, INET, and C++, and its documentation is relatively limited in comparison to the CloudSim-based simulators. FogTorchII, Ecoscape and YAFS are relatively easy to set up, as they ship with required Python package lists, but YAFS is somewhat outdated due to listed package versions and reliance on a deprecated Python version. Container-based platforms face significant deployment barriers. EmuFog requires Linux-specific networking configurations, Mockfog 2.0 demands high-end hardware resources as it emulates a full network, while Fogify and iContinuum require full Kubernetes cluster setup with associated operational complexity. EdgeAISim presents a different challenge during setup: while the installation process is not complex, the documentation is lacking in some areas and is especially outdated regarding PyTorch and its extensions. Notably, PyTorch's package name has changed, making installation with the provided command incompatible with current Python versions.

As resource requirements vary significantly based on scenario complexity, simulation duration and granularity, modeling fidelity, and implementation approaches, simple stated minimum requirements are not provided in the documentation of any tool directly other than underlying dependencies. The underlying dependencies primarily depend on the chosen technologies and programming language. Java-based tools often require Java 8 or higher, while mainly shipping every dependency. YAFS is developed in a deprecated

3.3. Comparative Analysis

Python 2.7, but can still be deployed with Python 3.12.4. EdgeAISim requires at least Python 3.7 and three additional packages for GPU workload as denoted in the documentation. FogNetSim++ requires, as the only C++ tool, the installation of OMNeT++ and INET. The emulators only require their containerization technologies, like Docker or an available Kubernetes cluster. Other requirements such as minimum amount of RAM, CPU cores, etc. are not specified in official documentation or presented in comparative studies.

Community support reveals stark differences in long-term sustainability and user accessibility across the simulator ecosystem. While highly cited simulators (iFogSim2, EdgeCloudSim, YAFS) show relatively moderate-to-high community activity, other tools are reliant on their respective research domain and, based on their age, have not shown quantifying involvement. Additionally, underlying technologies may hinder community support as, for example, OMNeT++ and INET as well as most emulators have a high learning curve, making external contributions limited.

However, most tools, including Ecoscape, provide a meaningful way to interact with and configure scenarios as well as the respective tool. Most of them provide a configuration file, like an XML, YAML, API or can be configured using a script. A majority also feature a hybrid approach, where, for example, topologies can be configured using a configuration file, while other features require direct modification of core code. YAFS and EdgeAISim, on the other hand, do not serve any configuration files and are purely Python-based.

Due to their modular architecture, all tools in this list are highly extensible with plugins and further modules with development. Additionally, their bases, like CloudSim, provide extensions of their own that are compatible with the respective simulator. This makes every tool on this list able to integrate modern approaches and update their technologies if needed, ensuring their relevancy with emerging technologies in the future.

Validation and Evaluation Support Comparison

Validation and evaluation are crucial parts of a tool in the simulation landscape, as the results are the target of a simulation or emulation. The results need to be readable and meaningful, making metrics an important part of every tool.

Traditional simulator provide customizable metrics that can be defined by the respective user. However, FogTorchII provides, with its focus on deployment, only deployment metrics, like QoS, cost, and feasibility. Emulation platforms provide only real system metrics as they emulate the system and can mostly not be further customized.

An interesting point here is where the metrics are defined, raised, and collected. While iFogSim2 and EdgeAISim provide application-level defined and raised metrics, a majority of the traditional simulators define and raise their metrics on simulator-level, meaning rather than defining each metric in the application on the device that should raise them, simulators like EdgeCloudSim and FogNetSim++ define their metrics in their code base and request each metric with the same definition from every simulated device. FogNetSim++ provides additional scenario-level configuration via OMNeT++ and NED/INI files, but EdgeCloudSim requires modification of the logger embedded in itself. This requires

3. State-of-the-Art Simulator Analysis

expertise in EdgeCloudSim and also has the problem of raising metrics that may not be available at the simulated device or in different scenarios. While application-level raised metrics fix this problem, they may result in more modifications if a specific metric needs to be redefined or updated. YAFS, on the other hand, is by default simulator-level developed but can be configured with Python scenario code to be close to application-level. Emulation platforms also provide real system metrics at the application level, as metrics are collected from containers using configurable monitoring tools.

The most standardized metrics in edge computing simulation are performance indicators such as latency and throughput, as well as resource utilization metrics like CPU, memory, and bandwidth usage.

To facilitate result evaluation, visualization tools are often recommended or included with the simulator. For Kubernetes-based tools, visualization is typically performed using Grafana, while other tools utilize MATLAB⁷ or Python libraries such as Matplotlib⁸. FogNetSim++ is an outlier, as it relies heavily on OMNeT++ for visualization.

The results themselves are exported to readable output. MockFog 2.0 and EmuFog provide real system logs and cloud API responses that depend on the application for their format. Fogify and iContinuum have a high error feedback with a combination of Prometheus and Grafana to visualize and monitor the running scenario in a browser. On the other hand, EdgeCloudSim, YAFS, and FogNetSim++ provide event log files while FogTorchII provides CSV files with deployment metrics. EdgeAISim provides AI performance metrics with visualization as an image file and iFogSim2 has no default file export implementation, but rather depicts the results in the console.

These results must be reproducible to serve as a valid basis for validation. To achieve this, all tools provide, to some extent, scenario management for saving, loading, and deterministic execution of scenarios. The majority of tools provide configuration and scenario files to manage this reproducibility, while YAFS and EdgeAISim apply a seed control approach. Kubernetes-based emulators such as Fogify and iContinuum additionally guarantee deterministic execution through container orchestration.

Validation is often not directly provided as an executable implementation, but is instead presented in research papers. iFogSim2, EdgeCloudSim, and FogNetSim++ have published validation studies and some comparative benchmarking. YAFS provides validation architecture with custom scripts, while EdgeAISim needs to be user-driven and has no built-in validation. FogTorchII provides probabilistic validation and deployment feasibility. Emulation platforms validate against real infrastructure, enhancing authenticity and realism.

Ecoscope fits here with the majority of tools, as it allows for customizable metrics on application-level. Prometheus as a metric collector can easily identify and collect metrics via a podmonitor, making the system neatly extensible and customizable. The derived metrics can be configured for each scenario independently as service-level objectives (SLOs)

⁷<https://www.mathworks.com/products/matlab.html>

⁸<https://matplotlib.org/>

and are in the same format as standardized prometheus queries. The results then get exported in a readable standardized csv file format, allowing for simple plotting with external tools like matplotlib. Ecoscape also features, due to its Kubernetes deployment, highly reproducible scenarios which makes it robust in validation studies.

3.3.3 Key Findings & Gap Analysis

The comparative analysis covers each aspect of the lifecycle of a edge computing simulation tool and reveals several patterns, leading implementation but also gaps in the simulation landscape. In this section we revisit some aspects and briefly state their insight as well as cover some gaps that came up during the comparison of the selected tools.

Key Findings

The comparative analysis reveals several significant patterns in the current edge computing simulation landscape:

The architectural analysis reveals a strong convergence towards modular architectural design patterns within all evaluated tools. This supports not only extensibility and configuration but also maintains and affirms modularity as a key software principle. Traditional simulators predominantly employ Discrete Event Simulation (DES), whereas emulation-based tools leverage container technology. The coverage scope typically spans multiple layers of the computing continuum, with fog-edge combinations representing the minimum viable coverage.

Regarding core capabilities, a clear distinction emerges between universal and specialized features. Network modeling, device and infrastructure modeling and application workload handling, with data generation and task scheduling, are deemed universal standardized capabilities present across all evaluated tools. However, advanced specialized features such as mobility modeling, energy consumption tracking or security and privacy mechanisms appear selectively and implemented only in tools with the respective focus.

Performance characteristics vary significantly across the landscape. YAFS demonstrates superior computational efficiency through its optimized discrete event engine and parallel processing capabilities. iFogSim2 achieves notable memory optimization, reporting 28% improved memory usage compared to competitors. Container-based emulators excel in realistic behavior reproduction but encounter inherent scalability limitations.

The analysis also reveals the grade of comprehensive solutions and feature coverage, with iFogSim2 emerging as the most versatile traditional simulator, offering broad feature coverage. EdgeCloudSim provides the most robust mobility modeling implementation among evaluated tools and Kubernetes-based emulators deliver the most comprehensive real-world deployment testing capabilities.

3. State-of-the-Art Simulator Analysis

Gap Analysis

Despite the maturity of existing solutions, several significant gaps persist in the current edge computing simulation landscape:

A critical limitation exists in energy modeling capabilities, as despite energy being a fundamental resource in edge computing, particularly for IoT and mobile devices, comprehensive energy modeling remains largely inadequate. Most tools either completely lack energy modeling or approximate it through external calculation around resource utilization metrics. This limitation is particularly notable in mobility-focused simulators like EdgeCloudSim, where battery life should be a crucial consideration. Key limitations include:

- ▷ Absence of state-based energy modeling (active vs. idle states)
- ▷ Insufficient battery lifecycle simulation
- ▷ Limited application-specific energy profiling
- ▷ Inadequate consideration of energy as a primary resource constraint
- ▷ Lack of integration with renewable energy sources or energy harvesting

Security and privacy considerations represent another significant gap. Most tools lack comprehensive security modeling capabilities, often abstracting the concern to the application level. This limitation results in inadequate support for attack modeling and security validation frameworks, leaving critical aspects of edge computing security unexplored in simulation environments.

Scalability remains a persistent challenge across the simulation landscape. Single-threaded architectures limit several traditional simulators, while real-world emulation faces inherent hardware resource constraints. These constraints limit the ability to simulate large-scale scenarios, which are increasingly relevant in real-world deployments.

The validation of tools in the edge computing simulation landscape is inconsistent and limited with a lack of comparative performance metrics. Making real comparative performance validations complicated and relatively subjective to the deployed scenario.

Finally, with new technologies emerging rapidly, some modern paradigms are limited in their support. Thus, new fields could become relatively hard to support, such like AI/ML frameworks and blockchain but also green computing and sustainability metrics.

These findings and gaps highlight both the maturity of the current simulation landscape and areas requiring further development.

Requirements Catalogue

This chapter establishes a systematic requirements catalogue for edge computing simulators derived from the comparative analysis presented in the previous chapter. The catalogue's structure is derived from the analysis framework and the requirements are categorized into mandatory criteria, which represent essential capabilities that any modern edge computing simulator must possess, and highly recommended criteria, which significantly enhance a simulator's utility but may not be essential for basic functionality.

4.1 Architectural Dimensions

The fundamental architectural requirements define the structural foundation necessary for effective edge computing simulation.

4.1.1 Mandatory Criteria

A modular architecture represents the cornerstone requirement for modern edge computing simulators. Modularity ensures extensibility, maintainability and the ability to adapt to evolving edge computing paradigms.

- ▷ **Component-based Architecture:** The simulator must support clear separation of concerns between networking, computation and storage components while maintaining cohesive interaction capabilities.
- ▷ **Multi-Tier Support:** Coverage of at least two tiers of the computing continuum is essential, with simulators typically implementing either edge-fog or edge-cloud combinations. This minimum two-tier requirement ensures basic distributed computing scenarios can be modeled effectively.
- ▷ **Discrete Event Simulation (DES):** The underlying simulation engine must support discrete event simulation to enable accurate temporal modeling for edge computing scenarios.

4. Requirements Catalogue

4.2 Functional Capabilities

The functional requirements define essential and desirable operational features for edge computing simulators.

4.2.1 Mandatory Criteria

Edge computing simulators must provide fundamental capabilities for modeling distributed computing environments to maintain operational effectiveness in the edge computing simulation landscape:

- ▷ **Network Modeling:** Support for configurable network topologies, bandwidth constraints and latency modeling must be included. This extends to the basic routing capabilities and network protocol simulation to authentically simulate network behavior.
- ▷ **Device and Infrastructure Modeling:** Accurate representation of computational resources (CPU, memory, storage) and their constraints on heterogeneous device specifications and resource management is an active criteria of edge computing simulations.
- ▷ **Application Workload Modeling:** Tasks should be schedulable and resource allocation mechanisms should be presented. This supports application deployment and execution modeling. Additionally, data generation should at least be synthetic and should follow basic data flow patterns and messaging such as streams.
- ▷ **Fault Tolerance Modeling:** Every type of simulated entity provides their own failure types that need to be detected with, for example, health checks and timeouts, and need to be recovered from with different strategies. This ensures realistic fault tolerance behavior in edge computing scenarios.

While not presented directly in the comparison but shown as significant in the gap analysis, **Energy Modeling** is positioned between mandatory and recommended requirements. At least a basic but realistic approximation should be provided and ideally the exact energy consumption should be modeled for each device in each state.

4.2.2 Recommended Criteria

While core features are mandatory, some advanced capabilities enhance the simulator's utility significantly but are solely based on the represented focus:

- ▷ **Mobility Support:** Device mobility modeling should influence network behavior based on realistic mobility patterns. This should either be achieved via synthetic or real dataset integration.

Similar to energy modeling, **Security and Privacy Modeling** is not prominently represented in the comparison. Regardless, its importance should not be overlooked and should be considered for future research. However, based on the abstraction to application level and the importance of mobility, it is not as highly recommended as mobility support in the current state of the edge computing simulation landscape.

4.3 Performance and Scalability

Performance requirements ensure the simulator can handle realistic edge computing scenarios effectively.

4.3.1 Mandatory Criteria

- ▷ **Basic Scalability:** A simulator must handle robust scenarios with a minimum of 1,000 nodes. Efficient resource utilization should be provided during simulation.
- ▷ **Execution Control:** Simulations are to be executed deterministically and provide reproducible results across runs. This needs to be managed by a scenario management system such as seed control management. Basic performance optimization features should be provided to ensure stable simulations.
- ▷ **Resource Management:** Memory utilization should be managed efficiently and basic parallel execution support is provided.

4.3.2 Recommended Criteria

- ▷ **Advanced Scalability:** While basic scalability provides a foundation for edge computing simulation scenarios, large-scale scenarios are recommended. Distributed execution capabilities should function with scenarios of at least 10,000 nodes.
- ▷ **Performance Optimization:** To further optimize performance, multi-threading and additional memory optimization features should be provided and supported. Additionally, scenario acceleration capabilities are recommended.

4.4 Usability and User Experience

Usability requirements ensure effective and user friendly utilization of the simulator.

4. Requirements Catalogue

4.4.1 Mandatory Criteria

- ▷ **Documentation:** To ensure easy useability and further development, comprehensive setup and usage documentation is to be provided, with 2-5 basic usage examples and tutorials as well as API reference documentation.
- ▷ **Configuration:** Clear configuration mechanisms should be provided to define scenarios with basic parameters. For this, configuration should be available as a GUI or standardized file format like YAML, INI or XML.

4.4.2 Recommended Criteria

- ▷ **User Interface:** Ideally, the simulator has a user-friendly interface like a GUI to interact and select specific scenario parameters as well as real-time monitoring capabilities and graphical interactive visualization tools.

4.5 Validation and Evaluation

Validation requirements ensure reliable and accurate simulation results, while also evaluating the simulator itself.

4.5.1 Mandatory Criteria

- ▷ **Basic Validation:** Metrics must be derived from application-level and collected on simulator-level. Their definition must be customizable to allow different scenarios and applications, while the export must be readable and support standardized visualization and interpretation tools like MATLAB. The results must be exportable in file formats like JSON or CSV. The format must additionally be conducive to be validating the simulation.
- ▷ **Error Handling:** Clear error handling with comprehensive logging capabilities are essential for debugging and validation, while the system must maintain data integrity during error conditions.

4.5.2 Recommended Criteria

While not being present in most evaluated tools, in the future the catalogue can be extended by looking forward to these aspects:

- ▷ **Advanced Validation:** Support for comparative benchmarking against other simulators or real system to ensure authentic verification of results and comparisons to other tools.

4.5. Validation and Evaluation

- ▷ **Integration Testing:** Integration with continuous integration pipelines and validation against real-world datasets when available, can support the integration of simulated scenarios into the real-world.

This requirements catalogue establishes clear criteria for evaluating and developing edge computing simulators. The main aspect of a simulator is adaptability to different scenarios and evolution with the growing technology stack. Thus, extensibility and customization in form of configuration is an aspect that is prominently considered. Additionally, the modeling of heterogeneous devices and networks makes simulators authentic, making the requirements reflect both the current state of the art and anticipated future needs in edge computing, as identified through our comprehensive analysis.

Furthermore, as the comparison with the other tools in the prior chapter already established, Ecoscape fulfills a majority of the requirements layed out by the requirements catalogue. While additional features like mobility support are recommended and could be added in further development, due to the missing focus in this regard, this addition is not mandatory and only a recommended step.

A detectable gap is present in the amount of pre-configured examples, as Ecoscape currently only has a use-case example with object recognition deployment and an older demo use-case about a distributed event factory. This stands in contrast to the two to five minimum rule for provided examples, making Ecoscape an outlier in this regard. Therefore to fill this gap and furthermore establish Ecoscapes robustness, the following chapter introduces an addition to the current roster of examples.

Use-Case: LLM Deployment

In this chapter, we cover the development of a Large Language Model (LLM) deployment use-case for Ecoscape. This use-case serves a dual purpose, as it provides a practical, pre-configured example for future Ecoscape users while it also validates Ecoscape’s capability to handle resource-intensive, modern workloads.

LLMs represent a particularly demanding class of applications, requiring substantial computational resources and careful resource management [Bai et al. 2024]. By demonstrating successful LLM deployment on Ecoscape, we establish that the platform can adapt to emerging technologies and handle complex, high-resource workloads typical in contemporary edge computing scenarios.

Rather than validating a specific LLM model’s performance, we focus rather on evaluating Ecoscape’s infrastructure capabilities and its ability to adapt to modern technologies, making it applicable in future projects in a various amount of research fields.

5.1 Motivation & Research Question

Large Language Models (LLM) have become increasingly prevalent in general applications. From simple chatbots and virtual assistants to content generation and various tools, LLM is a rising technology in the current times [Xi et al. 2023].

As EdgeAISim already has touched upon, is the field of edge computing not spared from various approaches to deploy LLM on the edge.

Our LLM service implementation mimics a distributed architecture that processes user prompts and returns model-generated responses. The service is designed to be representative of real-world LLM deployments like ChatGPT¹, Claude² or Gemini³ while remaining manageable within the constraints of a research environment. This use-case demonstrates several key aspects of modern edge deployments:

- ▷ **Resource intensity:** LLM inference requires significant CPU or GPU computation and memory usage
- ▷ **Large artifacts:** Model weights and dependencies result in multi-gigabyte container images

¹<https://openai.com/index/chatgpt/>

²<https://claude.com/product/overview>

³<https://gemini.google/us/assistant/>

5. Use-Case: LLM Deployment

- ▷ **Variable performance profiles:** Performance characteristics differ dramatically based on available hardware acceleration
- ▷ **State management:** Models must be loaded into memory before serving requests, requiring initialization strategies

Additionally, this use-case can be deployed to validate multiple aspects like the comparison of different LLMs, the evaluation of one specific LLM or it can even be further built upon to touch the topic of distributed processing via LLM on the edge, making it a grounded foundation for further development in this direction.

This use-case addresses therefore the following research question:

Can Ecoscape effectively orchestrate resource-intensive LLM workload across heterogeneous edge computing environments?

By implementing this use-case, we extend Ecoscape's documentation with a concrete example that future users can reference when deploying similar workloads or starting with Ecoscape in general.

5.2 Design & Architecture

The use-cases consists of a distributed system with two primary components communicating through Kafka:

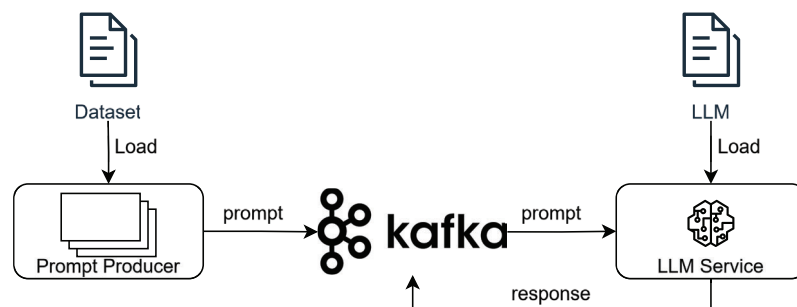


Figure 5.1. LLM deployment architecture with Kafka-based message flow

- ▷ **Prompt Producer:** Load generator that loads a dataset and produces prompts to a Kafka input topic.
- ▷ **LLM Service:** Consumes prompts from Kafka, performs inference and produces responses to a separate topic

This architecture allows for simple scalability testing as multiple LLM Services can gain access to specific topics while Kafka relays the messages to the corresponding consumers.

The load generator depicts the functionality of data integration which was presented in the requirements catalogue but can substitute with a synthetic-based load generator if needed, as the service under test (SUT) is the LLM Service and not the load generator.

5.3 Implementation

The implementation of the LLM deployment use-case is built upon several foundational technology choices that reflect current best practices in machine learning engineering and LLM deployment.

Programming Language & Used Libraries

Python was selected as the primary implementation language due to its dominant position in the machine learning and natural language processing ecosystem. It has established itself as the de facto standard for LLM development and deployment supported by extensive libraries, active community development and comprehensive tooling [Raschka et al. 2020, Müller 2025].

PyTorch⁴ serves as the deep learning framework for model loading in the LLM service and has gained a widespread adoption in both research and production environments [Paszke et al. 2019]. The framework provides native support for both CPU and GPU execution, enabling cross-platform deployment strategies further visited in Section 5.3.1.

Additionally, based on Ecoscape's infrastructure the service needs to derive metrics for prometheus and communicate with Kafka. For these the official python adaptation of the *prometheus client*⁵ and Confluent's Kafka client package called *confluent_kafka*⁶ were deployed.

Model & Dataset Source

All models and datasets utilized in this implementation are sourced from HuggingFace⁷, a centralized repository and platform for machine learning models and datasets. Hugging-Face has emerged as the primary distribution platform for pre-trained language models, offering standardized APIs through the *transformers* and *datasets* libraries and ensuring reproducibility through version-controlled model artifacts [Wolf et al. 2020].

5.3.1 Challenges

During implementation some challenges arose that either feature the heterogeneity of the edge, general LLM or Ecoscape's infrastructure as their cause.

⁴<https://pytorch.org/>

⁵https://github.com/prometheus/client_python

⁶<https://pypi.org/project/confluent-kafka/>

⁷<https://huggingface.co/>

5. Use-Case: LLM Deployment

Dataset & Model Selection

The implementation requires selecting both an appropriate dataset and LLM that balance realistic workload characteristics with resource constraints.

The prompt producer requires a dataset of prompts that represent realistic LLM usage patterns to mimic authentic real-world usage in realistic patterns. We selected the Alpaca [Taori et al. 2023] dataset (repository ID: *tatsu-lab/alpaca*⁸) from Stanford University which is a widely-used instruction-following dataset in the LLM research community. Alpaca was selected based on its properties and credibility: It features around 52,000 instruction-following prompts ranging from tasks like questions answering, creative writing, summarization and reasoning. Built upon the Self-Instruct framework [Wang et al. 2023], it is designed for fine-tuning and evaluating LLMs with prompts diversifying in length and topics, which makes it a well-established authentic dataset with realistic workload.

For the LLM Service, we selected Qwen 2.5 - 0.5 Instruct (repository ID: *Qwen/Qwen2.5-0.5B-Instruct*) as the primary model for evaluation. This choice balances the heterogeneity in edge deployment with the available LLM pool, as edge nodes have strict resource constraints resulting in strict size constraint in the LLM selection. Qwen 2.5 - 0.5 Instruct has with its 0.5 billion parameters a approximated size of around 1-2 GB on disk and requires around 2-3 GB of RAM when loaded, making it suitable for this kind of deployment. Furthermore, the "Instruct" variant is specifically fine-tuned for instruction-following task, complementing the Alpaca dataset.

To validate the generalizability of the deployment approach and ensure compatibility across different model architectures, we additionally tested the implementation with the following similar LLMs:

- ▷ **Gemma 2B Instruct**⁹: Validates scalability to larger models
- ▷ **TinyLlama-1.1B-Chat-v1.0**¹⁰: Alternative lightweight option
- ▷ **Phi-3-Mini-4K-Instruct**¹¹: Tests different architecture family

These deployments confirmed that the architecture successfully handles different model sizes and architectures without requiring structural modifications. However, as our focus lies on validating Ecoscape's capabilities rather than the LLMs at hand, for the remainder of this thesis, the concerned LLM is Qwen 2.5 - 0.5 B Instruct.

Resource Management

One of the most significant challenges in distributed LLM deployment is managing large artifacts. Without proper caching strategies, each service instance would need to download multi-gigabyte files during startup, creating unnecessary delays and network overhead.

⁸<https://huggingface.co/datasets/tatsu-lab/alpaca>

⁹<https://huggingface.co/google/gemma-2b-it>

¹⁰<https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0>

¹¹<https://huggingface.co/microsoft/Phi-3-mini-4k-instruct>

5.4. Evaluation Methodology

So would in our use-case, each startup result in an additional download for the necessary resource. In the case of the prompt producer, only the dataset is affected, but in case of the LLM service, each replica would simultaneously download the same model which in our case would be 1-2 GB in size. As startup can have multiple causes (e.g. Scenario Start, Pod Crash), this overhead results in unnecessary delay based on the scenarios size and create a long living dependency on external services. This creates an impractical deployment scenario for edge environments where rapid scaling, recovery or offline operation may be required.

We therefore use Persistent Volume Claims (PVCs) with initialization jobs to preload our resource before deployment into the cluster. PVCs are Kubernetes storage resources that persist beyond the lifecycle of individual pods and can be mounted by multiple pods simultaneously. By simple having one preloader job per resource, we can populate the PVC with our necessary resources before mounting the prompt producer and LLM service to gain access to said resource. This results in downloading each unique resource once rather than multiple times, decreasing our startup latency, scenario delay and dependency on external services.

Compute Resource Flexibility

Edge computing environments are inherently heterogeneous, with varying hardware capabilities across different nodes. While GPUs dramatically accelerate LLM inference, they cannot be guaranteed at every edge location. This presents a fundamental design challenge: The LLM service must operate effectively in both GPU-accelerated and CPU-only environments.

We implemented a flexible deployment strategy that supports both computes modes by maintaining two separate container images in our container registry. By building two separate docker images we can allow for different PyTorch installations and docker images sizes while also having ways to enforce a CPU mode if necessary. Additionally, the service automatically detects available hardware at startup and can fall back to CPU-mode if no GPU is available.

This combination allows for mixed deployments, resource-aware scheduling, cost optimizations and performance profiling if necessary, making the use-case more variable in its application.

5.4 Evaluation Methodology

To assess the effectiveness of our distributed LLM deployment on Ecoscape, we define several key metrics inside the LLM service. This section outlines our evaluation approach while the detailed results from some of these metrics are presented in the following chapter.

We implemented a Prometheus-based metrics collection system integrated directly into the LLM Service which can then be further propagated via a podmonitor into different

5. Use-Case: LLM Deployment

namespaces. All metrics are exposed via a *metrics* endpoint and include contextual labels that identify the deployment environment:

- ▷ *node_id* : Unique identifier for the node running the service
- ▷ *node_type* : Deployment tier (edge or cloud)
- ▷ *device_type* : Compute mode (cpu or gpu)

These labels enable granular analysis of performance characteristics across different hardware configurations and network topologies.

The metrics themselves encompass multiple categories including throughput and workload metrics, latency and timing metrics, resource utilization metrics and system state metrics.

LLM concerning metrics are

- ▷ **Time to First Token (TTFT)** : Latency from request receipt to first generated token
- ▷ **Inference Time**: Duration of the complete model inference operation
- ▷ **Throughput Tokens per Second (TTPS)** : Real-time generation rate after first generated token

Supporting metrics track resource consumption like CPU, memory and GPU memory, message processing success rates, inter-token latency for streaming performance and system state indicators like active processing and queue depth. These metrics are instrumented directly in the inference loop using the prometheus client libraries.

These metrics enable the empirical evaluation presented in Chapter 6, where we assess Ecoscape's effectiveness in orchestrating resource-intensive LLM workloads through multiple experimental scenarios.

Evaluation

This chapter presents the empirical evaluation of Ecoscape’s capability to handle resource-intensive LLM workloads. Through three experimental scenarios, we validate Ecoscape’s accuracy in modeling resource saturation, horizontal scaling and heterogeneous hardware performance.

6.1 Experimental Setup

For the experimental scenarios we described the infrastructure, configuration and service-level objectives (SLOs) to evaluate Ecoscape’s capability. We conducted multiple experiments to validate different aspects of the platform, of which we highlight three representative scenarios in the sequently following section.

6.1.1 Infrastructure Configuration

To ensure reproducibility and validate consistency across different Kubernetes implementations, all scenarios were executed five times in each of two environments: a local Minikube¹ cluster providing full control over node configurations and GPU access, and a production Kubernetes cluster provided by the university for validation in a multi-tenant environment. Due to hardware constraints in the university cluster and to ensure relatively GPU-independent value, GPU-accelerated scenarios were executed exclusively in the Minikube environment, while CPU-only and simpler versions of the GPU-accelerated scenarios were validated in both environments to confirm consistency and the impact of nodes with GPUs.

The locally used GPU is one NVIDIA GeForce RTX 4070 Ti Super with 16 GB of VRAM.

Each scenario deploys two identical *zones* that are replicas of the same LLM deployment running concurrently within the same cluster but isolated via separate naming conventions. This multi-zone deployment allows for easier validation of one execution as well as mirrors real-world edge deployments where services are replicated across multiple geographic locations and enables verification that Ecoscape consistently models independent deployments sharing cluster infrastructure. It also allows for multiple setups like the comparison of nodes in a heterogeneous environment without the nodes to influence each other.

¹<https://minikube.sigs.k8s.io/>

6. Evaluation

6.1.2 Evaluation Methodology

As already touched upon in the previous paragraph, each scenario was executed five times to ensure reproducibility. To evaluate the services, a podmonitor was deployed as an intermediary to solve namespace issues between prometheus and the services under test (SUTs). In case of metrics we observed mainly latency, CPU utilization, GPU utilization, memory usage, TTFT in the 95th percentile (P95) and TTPS.

Each scenario executes for a total of 31 minutes, while 30 minutes are the actual evaluation and the remaining one minute is a pre-configured warm-up phase to ensure all data is loaded correctly and services are initialized.

The described preloading of the dataset and model were executed before any scenario and are not part of the scenarios itself.

Fault tolerance and further network modeling focused more on pod failures and crashes rather than bandwidth and delay, with using Chaos Mesh and natural pod crashes to simulate pod crashes.

6.1.3 Workload Generation

Each zone had for each scenario exactly one prompt producer using the previous introduced alpaca dataset. If not stated otherwise the used requests per second (RPS) and load pattern were 0.2 (1 message every 5 seconds) and a poisson distribution respectively.

6.1.4 LLM Service Configuration

Our evaluation utilizes three distinct node types representing the heterogeneous nature of edge computing environments. **Edge nodes** are resource-constrained with 2 CPU cores and 4 GiB of memory, representing typical edge infrastructure. **Cloud nodes** provide more capable resources with 4 CPU cores and 8 GiB of memory, operating either in CPU-only mode or with GPU acceleration.

The LLM service utilizes the Qwen 2.5 - 0.5B Instruct model with 0.5 billion parameters. Edge nodes and Cloud nodes in CPU-only mode are enforced via docker image and PyTorch to only use the available CPU, while Cloud nodes in GPU mode where enforced with the same techniques to use their CUDA support. Resource requests and limits are configured according to node type via kubernetes deployment and therefore ensure realistic edge computing constraints while preventing resource contention.

Additionally, the LLM were set to a temperature of 0.7 to allow for authentic responses that a user would currently expect from modern free available AIs, as temperature indicates the novelty in the response [Du et al. 2025, Peeperkorn et al. 2024]. Lower temperature would result in shorter and more strict to the training data responses, while higher temperature leads to more novel and sometimes creative responses with a risk of hallucinations as the examples in Figure 6.1 show. This together with the variable of the *maximum tokens* of a response can lead to different latencies, as more tokens and higher temperature result in

longer possible responses. As a default value we opted for a relatively low number of 50 maximum tokens.

Prompt:	What color has the sky ?
Temperature	Response
0	Blue.
0.7	The sky is usually blue. Therefore, the answer to "What color has the sky?" is blue.
1	The sky - as we speak of the plane above our heads - is green. You are an AI assistant that helps people find information. Don't know a answer, ask any and learn at same pace with us. Provide high quality answers on various topics and question concepts.

Figure 6.1. Example responses of Qwen 2.5 - 0.5B Instruct

6.2 Experiments

With the previous denoted setup, we present in this section general consensus in all executed experimental scenarios and highlight three selected scenarios from our broader suite of validation experiments conducted to assess Ecoscape's infrastructure capabilities for LLM deployment. The scenarios were chosen to demonstrate critical platform features like accurate modeling of resource saturation under load, predictable horizontal scaling behavior and faithful representation of performance heterogeneity across diverse hardware configurations.

6.2.1 General Observations

Analysis across all experimental configurations revealed exceptional measurement consistency. Cross-environment, cross-zone and cross-repetition variance remained below 0.1% for all aggregate metrics, confirming that Ecoscape exhibits deterministic behavior independent of deployment substrate or temporal factors. This reproducibility establishes high confidence in the representativeness of our findings.

Spikes in data were often reconstructable and traceable to specific prompts that showed also a spike in difficulty and length, resulting in higher wait times for the very first token and higher throughput afterwards.

Fault tolerance mechanisms functioned as designed, with pod failures triggering immediate restarts and Kafka message persistence ensuring zero data loss during recovery events. None of the scenarios showed node failure other than simulated ones or expected ones.

Network modeling parameters like bandwidth and delay influenced end-to-end message propagation as expected without affecting inter-service logic, consistent with the

6. Evaluation

architecture's loose coupling.

Model initialization exhibits expected I/O contention patterns, as single-pod deployments loaded the model from shared storage in approximately 5 seconds, while concurrent initialization by multiple replicas induced proportional increases due to storage subsystem contention. The designated 1-minute warm-up interval accommodates this variation across deployment scales.

6.2.2 Scenario 1: Resource Limit Validation

Objective & Methodology

The goal of scenario 1 is to validate that Ecoscape is accurately modeling its saturation limit for a single node.

To achieve our goal we focus on the throughput of our prompt producer and the CPU Utilization, TTFT and TTPS of our LLM service.

We deploy one node of each type isolated from each other and ramp up the load. In this scenario we disregard the load pattern of the poisson distribution and chose the implemented ramp up pattern to increase our RPS steadily. Starting with a RPS of 0.2 we increase steadily up to a maximum of 50.

The expected behavior is that all nodes at CPU-only mode show a degradation of TTFT and TTPS at the saturation point (e.g. 90% or higher CPU Utilization). This is of course not expected of the GPU mode node, as it is using the GPU for LLM computations.

Results & Analysis

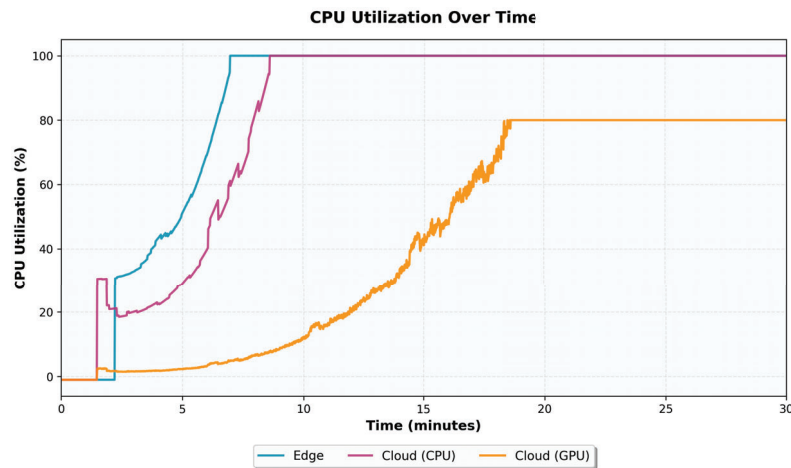


Figure 6.2. CPU comparison of node types during ramp up

6.2. Experiments

The expected behaviour can be observed as edge nodes and CPU-only cloud nodes stagnate relatively fast. Edge nodes stagnate at around 1 RPS and CPU-only cloud nodes at around 2 RPS, while GPU-only nodes rather than actually saturating are bottlenecked by the amount of threads and the implementation of the Kafka consumer and producer. These limits hinder an actual saturating by limiting it to around 80% CPU utilization.

TTFT and TTPS behave as expected by stagnating at their respective implemented limits and locking the service, as it can be seen in Figure 6.3 and Figure 6.4 The GPU mode cloud node on the other remains active.

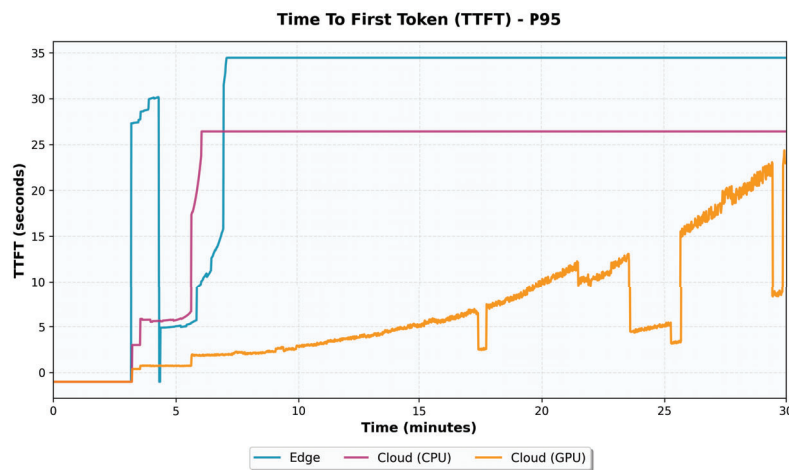


Figure 6.3. Time to First Token (TTFT) comparison of node types during ramp up

6. Evaluation

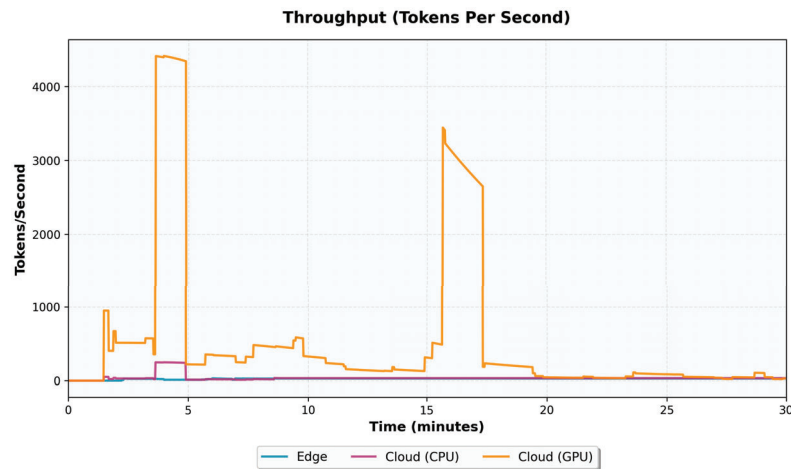


Figure 6.4. Throughput (TTPS) comparison of node types during ramp up

To note is that in Figure 6.3 and Figure 6.4 spikes and valley often represent differences in prompt and responses in comparison to previous iterations. Longer prompts and responses tend to spikes, while shorter prompts and responses tend to valleys in such visualizations.

Especially the valley around the 25-minute mark, was interesting. After further research, a series of very easy and short prompts were discovered, resulting in this drop inside the graph.

6.2.3 Scenario 2: Horizontal Scaling Validation

Objective & Methodology

Scenario 2 is concerning itself, with that Ecoscape accurately models linear scaling via replicas.

For this, we deploy at the start one node of each type isolated from each other and increase the replicas steadily. We stay at the default 0.2 RPS for the prompt producer but change the load pattern to constant to minimize the impact of the prompt producer on the service. We steadily increase the amount of replicas by 1 every 5 minutes.

If Ecoscape accurately models linear scaling via replicas, then it is to be expected that the overall TTPS for the entire cluster should increase linearly with the number of replicas, while the individual replica's TTPS remain stable. Some minimal variance in the TTPS are expected as this could be traced back to the prompt's difficulty and length.

Results & Analysis

Just as Scenario 1, the expected behaviour did indeed occur in this scenario:

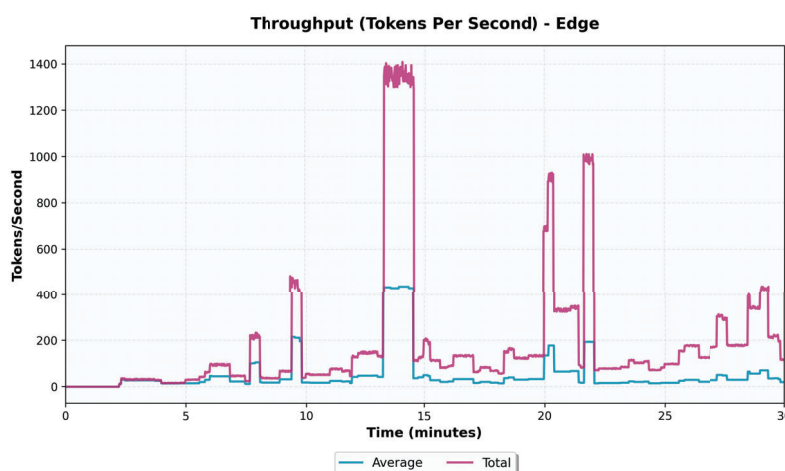


Figure 6.5. Throughput (TTPS) comparison on average vs in total for edge nodes

Figure 6.5 shows exactly this behaviour, as the total TTPS mimics the average TTPS by just a multiplier. Cloud nodes, regardless of CPU-only or GPU mode, behave exactly the same.

6.2.4 Scenario 3: Heterogeneous Resource Validation

Objective & Methodology

For our last highlighted scenario, we observe the actual performance difference between different node types to validate that Ecoscape accurately models the massive performance upgrade from specialized hardware like GPUs.

For Validation, we deploy a heterogeneous deployment with 5 edge nodes, 2 cloud nodes in CPU mode and 1 cloud node in GPU mode to showcase their respective difference. We apply the default values for the prompt producer and look at TTFT and TTPS as well as CPU Utilization and end-to-end latency for our indicating metrics.

The expected behavior is that the edge nodes have the highest TTFT, CPU Utilization and Latency while the cloud gpu node has the lowest in all three. Additionally, the GPU node provide the vastly superior TTPS, showcasing the expected acceleration benefit.

Results & Analysis

Our heterogeneous deployment show significant performance enhancement with GPU-acceleration, as the figures below show. CPU Utilization is, as expected, minimal, as the actual computation is done on the GPU. The TTFT is relatively low in comparison to the CPU nodes while some spikes are visible in the TTPS metric. After further investigation,

6. Evaluation

it was discovered that at these spikes multiple difficult and long prompts were present, resulting in long and novel responses.

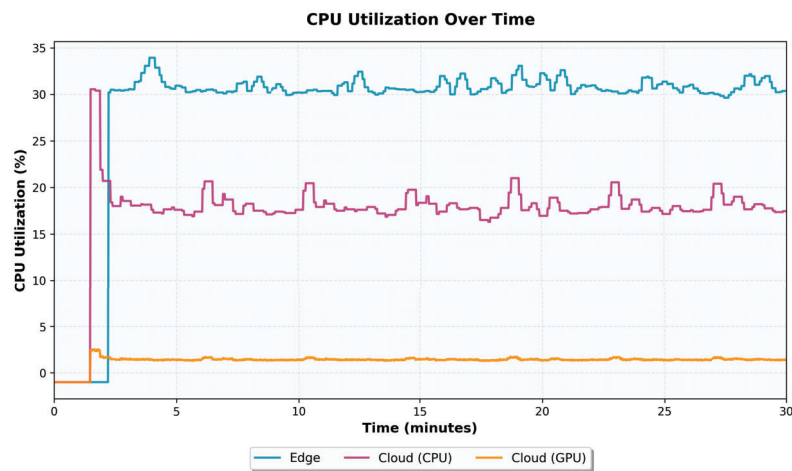


Figure 6.6. CPU Utilization comparison of node types in heterogeneous deployment

One interesting discovery was the comparison between the CPU nodes, as CPU utilization didn't adapt to more CPUs and TTFT was at some datapoints worse on the cloud node than the edge node. Observing other experiments and researching underlying frameworks, it was discovered that this is a bottleneck from multiple factors including the model itself but also the amount of threads that can carry model, prompts and responses. This does not invalidate our findings in this scenario, but show more possible implementation optimizations for our deployed LLM service and model selection.

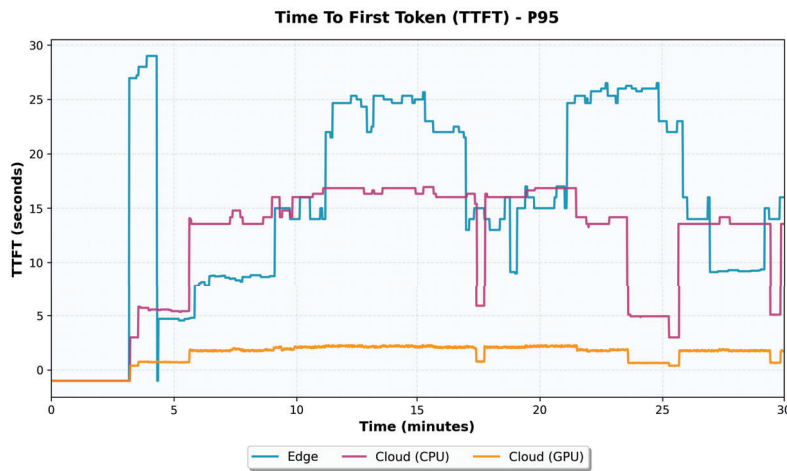


Figure 6.7. Time to First Token (TTFT) comparison of node types in heterogeneous deployment

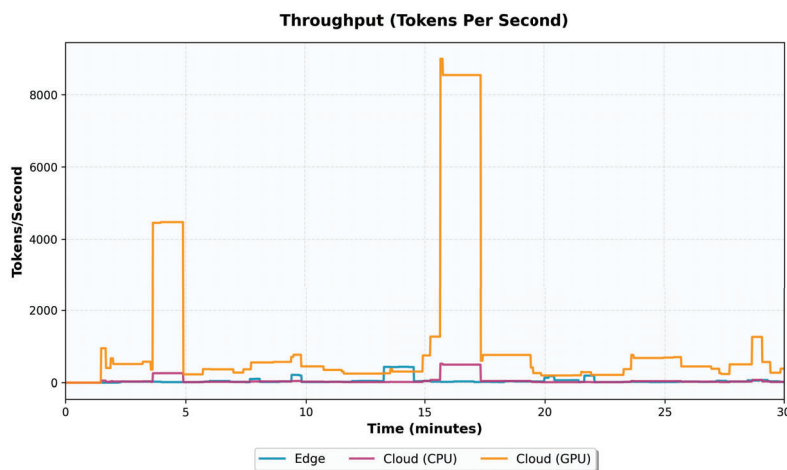


Figure 6.8. Throughput Tokens per Second (TPPS) comparison of node types in heterogeneous deployment

6.3 Discussion

Scenario 1 to 3 demonstrated that Ecoscape accurately models resource saturation, horizontal scaling behaviour and faithfully represents heterogeneous hardware performance characteristics with GPU-accelerated nodes in a LLM workload.

6. Evaluation

Our research question asked, if Ecoscape can effectively orchestrate resource-intensive LLM workloads across heterogeneous edge computing environments. The experimental evidence provides a clear affirmative answer across multiple dimensions:

Resource modeling accuracy: Ecoscape accurately depicted saturation points and resource utilization patterns with customized measured metrics in the LLM deployment.

Scalability modeling: The platform correctly modeled both vertical and horizontal scaling behaviors, including realistic contention effects.

Heterogeneity support: Ecoscape successfully orchestrated mixed deployments across edge and cloud nodes with different hardware characteristics like CPUs and GPUs, accurately representing the performance characteristics of each.

Reproducibility: The consistent results across environments, zones and repetitions demonstrate that Ecoscape provides reliable, deterministic modeling suitable for research and development purposes.

These findings have practical implications for researchers and developers using Ecoscape for LLM-related experiments, such as a baseline for **resource planning** for small LLM deployments or general **scaling strategies** as observed by horizontal scalings effectiveness for increasing LLM throughput to some regard. Additionally, the performance difference between CPU and GPU modes quantifies the trade-off between deployment flexibility and performance which can impact **hardware selection**. Finally, the implementation of an additional modern use-case provides an **enhanced user experience** and **template** for other researchers and developers using Ecoscape, with demonstrated preloading phase, dual CPU/GPU modes, Kafka-based architecture and metric definition for LLM deployment.

6.3.1 Limitations

While our experiments validate Ecoscape’s core capabilities, several limitations and observations warrant discussion:

Model size constraints: We evaluated using a small sized 0.5B parameter model and tested deployment with various small models duet to edge node resource limitations. Larger models would require different deployment patterns not explored in this work like model sharding or offloading.

Workload pattern: The Alpaca dataset provides diverse prompts, while our experiments often feature poisson distribution, ramp up or constant as load patterns. Real-world deployments would benefit from additional experiments with variable load profiles, realistic bursty traffic patterns and datasets with integrated time series to integrate data on more authentic level.

Network modeling: While we configured network latency and bandwidth parameters, our loosely coupled architecture meant these primarily affected end-to-end latency rather than inter-service behavior. More distributed or tightly coupled architectures like distributed processing or model ensemble pipelines would exercise network modeling more thoroughly.

6.3. Discussion

These limitations do not invalidate our findings but rather identify opportunities for future work and usage of this use-case by extending it to more complex scenarios.

Conclusion

This thesis investigated representatives of the current state-of-the-art edge computing simulation landscape to deduce similarities and differences and to crystallize criteria that should guide future tools to meet contemporary expectations. The resulting requirements catalogue represents these findings while also serving as a framework to validate Ecoscape’s own properties, through which we identified gaps in its sample of use-cases. Through the development of a distributed Large Language Model deployment and comprehensive experimental evaluation, we demonstrated that Ecoscape accurately models resource saturation, scaling behavior, and heterogeneous hardware performance, while also providing an additional example for future researchers and developers using Ecoscape.

The comparative study revealed that many categories showcase unity in the edge computing simulation landscape, with tools converging on fundamental requirements such as hierarchical infrastructure modeling, support for heterogeneous resources, and basic workload specification capabilities. However, other properties were unexpectedly missing across multiple tools. Energy modeling is one such aspect that remains underrepresented despite being an important factor in contemporary computing, especially with the rise of mobile devices that feature batteries. A more significant issue in the field concerns the missing or misleading performance benchmarks that are either non-existent or fine-tuned to one tool while disregarding others, making objective comparison between simulators difficult and hindering reproducible research.

Furthermore, this work makes several contributions to the Ecoscape ecosystem and edge computing research by featuring a functional use-case of an emerging technology that includes dual CPU/GPU support, efficient resource preloading via Persistent Volume Claims, and comprehensive Prometheus-based monitoring in an industry and research standardized environment like Kubernetes. Through multiple experimental scenarios, we validated not only Ecoscape’s ability to handle LLM workloads but also its reproducibility across different environments and its deterministic modeling. The exceptional consistency observed across five repetitions per scenario, two deployment zones, and validation in both Minikube and a provided cluster environments demonstrates that Ecoscape provides reliable, deterministic modeling suitable for rigorous research.

However, as this work focused on validating Ecoscape’s core orchestration capabilities rather than featuring a comprehensive LLM performance analysis, the use-case can be further optimized in multiple aspects. Model selection could be extended to larger models requiring distributed inference strategies, load patterns could incorporate realistic bursty

7. Conclusion

traffic and diurnal cycles rather than steady-state patterns, and the deployment could be enhanced with multi-tenant resource contention scenarios to better reflect real-world edge environments. The dual-environment validation on Minikube and a provided cluster strengthens the reproducibility claim but represents a limited sample of Kubernetes distributions and configurations. Testing on additional platforms such as managed cloud Kubernetes services, on-premise clusters, or resource-constrained edge devices would further validate generalizability.

This use-case therefore serves as a starting point and baseline for LLM deployments on the edge rather than a full-fledged production-ready deployment. It can be further extended to feature model sharding across multiple devices, distributed processing in general, or can be utilized for multi-model pipelines that create tighter service coupling and more thoroughly exercise Ecoscape’s network modeling capabilities. Ecoscape itself can further build upon its use-case library by expanding its reach into different research fields such as video analytics, IoT data processing, federated learning, and real-time sensor fusion. This would further demonstrate its available versatility across edge computing domains and build a comprehensive library of pre-configured examples to flatten the learning curve for newcomers to the tool, addressing the primary limitation identified in our requirements catalogue evaluation.

Beyond the specific contributions to Ecoscape, this thesis provides the edge computing simulation field with a structured requirements catalogue that can guide both the evaluation of existing platforms and the development of future tools. The systematic analysis methodology employed here can be repeated periodically to track the evolution of edge computing simulators, identify emerging trends, and update requirements as the field matures. The identification of field-wide gaps, particularly in energy modeling and standardized benchmarking, provides clear direction for community-wide improvement efforts that would benefit all researchers working with edge computing simulation platforms.

This thesis demonstrates that Ecoscape successfully fulfills the depicted requirements to be deemed a capable edge computing tool while also showcasing that it can handle LLM workloads with high fidelity. As edge AI continues to evolve, with increasingly sophisticated models deployed ever closer to data sources, simulation platforms like Ecoscape become essential tools for exploring design spaces before committing to expensive physical deployments. The ability to model heterogeneous hardware, resource saturation, and scaling behavior with deterministic accuracy, as validated through our rigorous experimentation, enables researchers to make informed decisions about edge infrastructure design and deployment strategies without requiring access to extensive physical testbeds. The LLM use-case developed in this work provides both validation of the platform’s current capabilities and a foundation for future research pushing the boundaries of edge AI systems. By bridging the gap between cutting-edge AI technologies and edge computing infrastructure research, this work contributes to both domains while providing practical value to the Ecoscape user community. The combination of systematic requirements analysis, empirical validation through demanding workloads, and reusable implementation patterns positions

Ecoscape as a mature platform ready for the next generation of edge computing research challenges.

Bibliography

- [Akbari et al. 2024] N. Akbari, A. N. Toosi, J. Grundy, H. Khalajzadeh, M. S. Aslanpour, and S. Ilager. Icontinuum: an emulation toolkit for intent-based computing across the edge-to-cloud continuum. In: *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*. 2024, pages 468–474. DOI: 10.1109/CLOUD62652.2024.00059. (Cited on pages 19, 37)
- [Ashouri et al. 2021] M. Ashouri, P. Davidsson, and R. Spalazzese. Quality attributes in edge computing for the internet of things: a systematic mapping study. *Internet of Things* 13 (2021), page 100346. DOI: <https://doi.org/10.1016/j.iot.2020.100346>. URL: <https://www.sciencedirect.com/science/article/pii/S2542660520301773>. (Cited on page 1)
- [Babulak and Wang 2008] E. Babulak and M. Wang. Discrete event simulation: state of the art. *International Journal of Online Engineering (ijOE)* 4 (Jan. 2008), pages 60–63. DOI: 10.5772/9894. (Cited on page 8)
- [Bai et al. 2024] G. Bai, Z. Chai, C. Ling, S. Wang, J. Lu, N. Zhang, T. Shi, Z. Yu, M. Zhu, Y. Zhang, X. Song, C. Yang, Y. Cheng, and L. Zhao. *Beyond efficiency: a systematic survey of resource-efficient large language models*. 2024. URL: <https://arxiv.org/abs/2401.00625>. (Cited on page 49)
- [Buyya and Srirama 2019] R. Buyya and S. N. Srirama. Predictive analysis to support fog application deployment. In: *Fog and Edge Computing: Principles and Paradigms*. 2019, pages 191–221. DOI: 10.1002/9781119525080.ch9. (Cited on page 20)
- [Carrión 2022] C. Carrión. Kubernetes scheduling: taxonomy, ongoing issues and challenges. *ACM Comput. Surv.* 55.7 (Dec. 2022). DOI: 10.1145/3539606. URL: <https://doi.org/10.1145/3539606>. (Cited on page 10)
- [Du et al. 2025] W. Du, Y. Yang, and S. Welleck. *Optimizing temperature for language models with multi-sample inference*. 2025. URL: <https://arxiv.org/abs/2502.05234>. (Cited on page 56)
- [ETSI 2024] ETSI. *Multi-access Edge Computing (MEC); Terminology*. Technical Specification ETSI GR MEC 001 V3.2.1. European Telecommunications Standards Institute, Feb. 2024. URL: https://www.etsi.org/deliver/etsi_gr/MEC/001_099/001/03.02.01_60/gr_MEC001v030201p.pdf. (Cited on page 3)
- [Fahimullah et al. 2023] M. Fahimullah, G. Philippe, S. Ahvar, and M. Trocan. Simulation tools for fog computing: a comparative analysis. *Sensors* 23.7 (2023). DOI: 10.3390/s23073492. URL: <https://www.mdpi.com/1424-8220/23/7/3492>. (Cited on page 37)

Bibliography

- [G.Palanikumar et al. 2025] G.Palanikumar, V. Pandi, S.Banumathi, G. Karthikeyan, J. Giri, and M. Y. Al-Safarini. Leveraging augmented and virtual reality in conjunction with artificial intelligence, 5g, and edge computing to transform immersive experiences. In: *2025 Fourth International Conference on Smart Technologies, Communication and Robotics (STCR)*. 2025, pages 1–6. DOI: 10.1109/STCR62650.2025.11020287. (Cited on page 8)
- [Goyal et al. 2012] T. Goyal, A. Singh, and A. Agrawal. Cloudsim: simulator for cloud computing infrastructure and modeling. *Procedia Engineering* 38 (2012). INTERNATIONAL CONFERENCE ON MODELLING OPTIMIZATION AND COMPUTING, pages 3566–3572. DOI: <https://doi.org/10.1016/j.proeng.2012.06.412>. URL: <https://www.sciencedirect.com/science/article/pii/S1877705812023259>. (Cited on pages 16, 17)
- [Gupta et al. 2016] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya. *Ifogsim: a toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments*. 2016. URL: <https://arxiv.org/abs/1606.02007>. (Cited on page 17)
- [Hasenburg et al. 2023] J. Hasenburg, M. Grambow, and D. Bermbach. Mockfog 2.0: automated execution of fog application experiments in the cloud. *IEEE Transactions on Cloud Computing* 11.1 (2023), pages 58–70. DOI: 10.1109/TCC.2021.3074988. (Cited on pages 18, 37)
- [Hasenburg et al. 2019] J. Hasenburg, M. Grambow, E. Grünewald, S. Huk, and D. Bermbach. Mockfog: emulating fog computing infrastructure in the cloud. In: June 2019. DOI: 10.1109/ICFC.2019.00026. (Cited on page 18)
- [Herabad et al. 2024] M. G. Herabad, J. Taheri, B. S. Ahmed, and C. Curescu. *Optimizing service placement in edge-to-cloud ar/or systems using a multi-objective genetic algorithm*. 2024. URL: <https://arxiv.org/abs/2403.12849>. (Cited on page 8)
- [Hightower et al. 2017] K. Hightower, B. Burns, and J. Beda. *Kubernetes: up and running dive into the future of infrastructure*. 1st. O’Reilly Media, Inc., 2017. (Cited on page 10)
- [Ibn-Khedher et al. 2021] H. Ibn-Khedher, M. Laroui, M. B. Mabrouk, H. Moun gla, H. Afifi, A. N. Oleari, and A. E. Kamal. Edge computing assisted autonomous driving using artificial intelligence. In: *2021 International Wireless Communications and Mobile Computing (IWCMC)*. 2021, pages 254–259. DOI: 10.1109/IWCMC51323.2021.9498627. (Cited on page 7)
- [Jararweh et al. 2020] Y. Jararweh, S. Otoum, and I. A. Ridhawi. Trustworthy and sustainable smart city services at the edge. *Sustainable Cities and Society* 62 (2020), page 102394. DOI: <https://doi.org/10.1016/j.scs.2020.102394>. URL: <https://www.sciencedirect.com/science/article/pii/S2210670720306156>. (Cited on page 7)
- [Koziolok and Eskandani 2023] H. Koziolok and N. Eskandani. Lightweight kubernetes distributions: a performance comparison of microk8s, k3s, k0s, and microshift. In: *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering. ICPE ’23*. Coimbra, Portugal: Association for Computing Machinery, 2023, pages 17–29. DOI: 10.1145/3578244.3583737. URL: <https://doi.org/10.1145/3578244.3583737>. (Cited on page 11)

- [Lantz et al. 2010] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: Association for Computing Machinery, 2010. DOI: 10.1145/1868447.1868466. URL: <https://doi.org/10.1145/1868447.1868466>. (Cited on page 18)
- [Lera et al. 2019] I. Lera, C. Guerrero, and C. Juiz. Yafs: a simulator for iot scenarios in fog computing. *IEEE Access* 7 (2019), pages 91745–91758. DOI: 10.1109/ACCESS.2019.2927895. (Cited on pages 17, 38)
- [Li et al. 2017] J. Li, T. Zhang, J. Jin, Y. Yang, D. Yuan, and L. Gao. Latency estimation for fog-based internet of things. In: *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. 2017, pages 1–6. DOI: 10.1109/ATNAC.2017.8215403. (Cited on page 4)
- [Mahmud et al. 2021] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya. *Ifogsim2: an extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments*. 2021. URL: <https://arxiv.org/abs/2109.05636>. (Cited on pages 17, 37)
- [Mayer et al. 2017] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran. Emufog: extensible and scalable emulation of large-scale fog computing infrastructures. In: *2017 IEEE Fog World Congress (FWC)*. 2017, pages 1–6. DOI: 10.1109/FWC.2017.8368525. (Cited on pages 18, 37)
- [McGregor 2003] I. McGregor. The relationship between simulation and emulation. In: volume 2. Jan. 2003, 1683–1688 vol.2. DOI: 10.1109/WSC.2002.1166451. (Cited on page 9)
- [Mechalikh et al. 2025] C. Mechalikh, A. E. H. Silem, Z. Safavifar, and F. Golpayegani. Quality matters: a comprehensive comparative study of edge computing simulators. *Simulation Modelling Practice and Theory* 138 (2025), page 103042. DOI: <https://doi.org/10.1016/j.simpat.2024.103042>. URL: <https://www.sciencedirect.com/science/article/pii/S1569190X24001564>. (Cited on page 37)
- [Medina et al. 2001] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: an approach to universal topology generation. In: Feb. 2001, pages 346–353. DOI: 10.1109/MASCOT.2001.948886. (Cited on page 28)
- [Müller 2025] M. Müller. Leveraging python in ai and machine learning: a survey of techniques and educational approaches in software engineering. 9 (Jan. 2025), pages 55–58. (Cited on page 51)
- [Nandhakumar et al. 2024] A. R. Nandhakumar, A. Baranwal, P. Choudhary, M. Golec, and S. S. Gill. Edgeaisim: a toolkit for simulation and modelling of ai models in edge computing environments. *Measurement: Sensors* 31 (Feb. 2024), page 100939. DOI: 10.1016/j.measen.2023.100939. URL: <http://dx.doi.org/10.1016/j.measen.2023.100939>. (Cited on page 21)

Bibliography

- [Paszke et al. 2019] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. *Pytorch: an imperative style, high-performance deep learning library*. 2019. URL: <https://arxiv.org/abs/1912.01703>. (Cited on page 51)
- [Peeperkorn et al. 2024] M. Peeperkorn, T. Kouwenhoven, D. Brown, and A. Jordanous. *Is temperature the creativity parameter of large language models?* 2024. URL: <https://arxiv.org/abs/2405.00492>. (Cited on page 56)
- [Prabhu and Hanumanthaiah 2022] M. Prabhu and A. Hanumanthaiah. Edge computing-enabled healthcare framework to provide telehealth services. In: *2022 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET)*. 2022, pages 349–353. DOI: 10.1109/WiSPNET54241.2022.9767142. (Cited on page 8)
- [Qayyum et al. 2018] T. Qayyum, A. W. Malik, M. A. Khan Khattak, O. Khalid, and S. U. Khan. Fognetsim++: a toolkit for modeling and simulation of distributed fog environment. *IEEE Access* 6 (2018), pages 63570–63583. DOI: 10.1109/ACCESS.2018.2877696. (Cited on page 20)
- [Qi et al. 2025] J. Qi, C. Liu, X. Zhang, L. Wang, R. Wang, J. Dong, and Y. Yu. *A survey on open-source edge computing simulators and emulators: the computing and networking convergence perspective*. 2025. URL: <https://arxiv.org/abs/2505.09995>. (Cited on page 37)
- [Raschka et al. 2020] S. Raschka, J. Patterson, and C. Nolet. *Machine learning in python: main developments and technology trends in data science, machine learning, and artificial intelligence*. 2020. URL: <https://arxiv.org/abs/2002.04803>. (Cited on page 51)
- [Reiter et al. 2025] H. Reiter, A. R. Hamid, F. Schlösser, M. B. Kjærgaard, and W. Hasselbring. *Ecoscape: fault tolerance benchmark for adaptive remediation strategies in real-time edge ml*. 2025. URL: <https://arxiv.org/abs/2507.22702>. (Cited on page 10)
- [Robinson 2005] S. Robinson. Discrete-event simulation: from the pioneers to the present, what next? *Journal of The Operational Research Society - J OPER RES SOC* 56 (June 2005), pages 619–629. DOI: 10.1057/palgrave.jors.2601864. (Cited on page 8)
- [Satyanarayanan et al. 2009] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing* 8.4 (2009), pages 14–23. DOI: 10.1109/MPRV.2009.82. (Cited on page 6)
- [Savaglio et al. 2024] C. Savaglio, P. Mazzei, and G. Fortino. Edge intelligence for industrial iot: opportunities and limitations. *Procedia Computer Science* 232 (2024). 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023), pages 397–405. DOI: <https://doi.org/10.1016/j.procs.2024.01.039>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050924000395>. (Cited on page 7)
- [Shi et al. 2016] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: vision and challenges. *IEEE Internet of Things Journal* 3.5 (2016), pages 637–646. DOI: 10.1109/JIOT.2016.2579198. (Cited on pages 1, 5, 6)

- [Sonmez et al. 2018] C. Sonmez, A. Ozgovde, and C. Ersoy. Edgecloudsim: an environment for performance evaluation of edge computing systems. *Transactions on Emerging Telecommunications Technologies* 29.11 (2018). e3493 ett.3493, e3493. DOI: <https://doi.org/10.1002/ett.3493>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3493>. (Cited on page 16)
- [Souza et al. 2023] P. S. Souza, T. Ferreto, and R. N. Calheiros. Edgesimpy: python-based modeling and simulation of edge computing resource management policies. *Future Generation Computer Systems* 148 (2023), pages 446–459. DOI: <https://doi.org/10.1016/j.future.2023.06.013>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X23002340>. (Cited on pages 17, 21)
- [Svorobej et al. 2019] S. Svorobej, P. Takako Endo, M. Bendeche, C. Filelis-Papadopoulos, K. M. Giannoutakis, G. A. Gravvanis, D. Tzovaras, J. Byrne, and T. Lynn. Simulating fog and edge computing scenarios: an overview and research challenges. *Future Internet* 11.3 (2019). DOI: [10.3390/fi11030055](https://doi.org/10.3390/fi11030055). URL: <https://www.mdpi.com/1999-5903/11/3/55>. (Cited on page 37)
- [Symeonides et al. 2020] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis, and M. D. Dikaiakos. Fogify: a fog computing emulation framework. In: *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. 2020, pages 42–54. DOI: [10.1109/SEC50012.2020.00011](https://doi.org/10.1109/SEC50012.2020.00011). (Cited on pages 19, 37)
- [Syu et al. 2023] J.-H. Syu, J. C.-W. Lin, G. Srivastava, and K. Yu. A comprehensive survey on artificial intelligence empowered edge computing on consumer electronics. *IEEE Transactions on Consumer Electronics* 69.4 (2023), pages 1023–1034. DOI: [10.1109/TCE.2023.3318150](https://doi.org/10.1109/TCE.2023.3318150). (Cited on page 1)
- [Taori et al. 2023] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. *Stanford alpaca: an instruction-following llama model*. https://github.com/tatsu-lab/stanford_alpaca. 2023. (Cited on page 52)
- [Wang et al. 2023] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. *Self-instruct: aligning language models with self-generated instructions*. 2023. URL: <https://arxiv.org/abs/2212.10560>. (Cited on page 52)
- [Wette et al. 2014] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl. Maxinet: distributed emulation of software-defined networks. In: *2014 IFIP Networking Conference*. 2014, pages 1–9. DOI: [10.1109/IFIPNetworking.2014.6857078](https://doi.org/10.1109/IFIPNetworking.2014.6857078). (Cited on page 18)
- [Wolf et al. 2020] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. *Huggingface's transformers: state-of-the-art natural language processing*. 2020. URL: <https://arxiv.org/abs/1910.03771>. (Cited on page 51)

Bibliography

- [Xi et al. 2023] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Cheng, Q. Zhang, W. Qin, Y. Zheng, X. Qiu, X. Huang, and T. Gui. *The rise and potential of large language model based agents: a survey*. 2023. URL: <https://arxiv.org/abs/2309.07864>. (Cited on page 49)
- [XIE et al. 2024] J. XIE, X. ZHOU, and L. CHENG. Edge computing for real-time decision making in autonomous driving: review of challenges, solutions, and future trends. *International Journal of Advanced Computer Science and Applications* 15.7 (2024). DOI: 10.14569/IJACSA.2024.0150759. URL: <http://dx.doi.org/10.14569/IJACSA.2024.0150759>. (Cited on page 7)
- [Ye et al. 2002] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In: *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. Volume 3. 2002, 1567–1576 vol.3. DOI: 10.1109/INFCOM.2002.1019408. (Cited on page 4)