National College of Ireland

Performance Optimization of Serverless Edge Computing for Machine Learning Workloads in Distributed Edge Environments

MSc Research Project
Cloud Computing

Akash Anil Sane
Student ID: 21220956

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akash Anil Sane
Student ID:	21220956
Programme:	Cloud Computing
Year:	2023/2024
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	31/01/2024
Project Title:	Performance Optimization of Serverless Edge Computing for Machine Learning Workloads in Distributed Edge Environments
Word Count:	6658
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Akash Anil Sane
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Performance Optimization of Serverless Edge Computing for Machine Learning Workloads in Distributed Edge Environments

Akash Anil Sane
21220956

Abstract

This research focuses on latency and resource optimisation issues in deploying machine learning applications in distributed environments using serverless edge computing. The increasing number of IoT devices requires a strong infrastructure that can effectively handle large amounts of data. The objective of the project is to enhance the scalability and response time of systems by breaking down machine learning applications into smaller, manageable serverless functions. The study leverages the CIFAR-10 and ImageNet200 datasets to assess and improve the system's performance. It demonstrates significant gains in handling initial startup processes and various workloads. This research enhances the understanding of serverless computing paradigms by providing a novel method that aligns with the latest progress in the field. Essentially, this research presents a framework that greatly simplifies the implementation of machine learning in edge environments, thereby addressing current business needs for responsiveness and efficiency. However, the study does not address long-term sustainability but informs about using a federated learning approach for serverless functions to enhance model accuracy, which uses large datasets, leaving room for future investigation.

Keywords: serverless computing, edge computing, machine learning, CIFAR-10, ImageNet200, resource utilization, scalability.

1 Introduction

Growth of the IoT resulted in a significant increase in data produced at the edge of the network, which poses distinct difficulties for data management and application implementation. The integration of machine learning (ML) applications into evolving edge computing systems has become crucial for analysing and acting upon the data in real-time. Conventional cloud computing infrastructures, although strong, can cause delays that may limit the required responsiveness for edge-based applications. Serverless computing has become a revolutionary method in this situation, providing benefits such as automated scaling, easier maintenance, and a pay-as-you-go pricing structure that guarantees cost savings and operational efficiency.

Although serverless computing is becoming more popular, it does have some downsides. Problems with persistent workloads, increased operational expenses because of

cloud service architecture, performance gaps in edge ML applications, and latency during "cold start" times are critical. The urgent requirement for optimised computing paradigms to meet the ever-changing demands of edge ML applications is highlighted by these concerns. According to Shafiei et al. (2022), the phrase "serverless" is frequently used to misunderstand the real-world functioning of such designs. It does not indicate that there are no servers present; rather, it abstracts server management from the end-users. There are two different methods that have become popular in serverless computing, which is Function-as-a-Service and Backend-as-a-Service. BaaS systems, such as Firebase and AWS Amplify, are designed to save developers from the complexities of server-side operations. On the other hand, FaaS solutions, such as AWS Lambda, provide a more granular degree of execution control, enabling code to run in response to events on a function-by-function basis.

Based on the existing literature and practical implementations of serverless computing for machine learning workloads in edge environments, the objective of this study is to address the identified limitations. In a serverless environment, the objective is to discover and evaluate innovative metrics and approaches that have the potential to evaluate and enhance machine learning model deployment. To achieve the best possible balance between performance and functionality, this project has investigated important metrics like scalability, resource utilisation, and response time.

This study builds upon the work of author Bac et al. (2022), who presented a serverless framework for edge ML applications, and proposes a more sophisticated technique, building on their initial assumptions. To improve efficiency and study the scalability aspect, the research has looked into breaking ML models into smaller serverless functions. This study aims to address the problems and limitations pointed out in previous research by adding new metrics like CIFAR-10 and ImageNet200, looking into optimisation methods, and testing how effectively they work in the framework of serverless edge computing. This research aims to significantly contribute to the growing field of serverless computing, specifically in relation to edge ML workloads, by providing a thorough examination of these methodologies and carefully validating them.

1.1 Research Question

- **By Incorporating additional metrics such as CIFAR-10 and ImageNet200 and dividing ML applications into smaller serverless functions, how to enhance the performance of machine learning workloads in distributed serverless edge computing environments?**

1.2 Objective

The primary objective of this project is to investigate techniques for improving the performance of machine learning workloads in distributed serverless edge computing systems. This will be accomplished by implementing the following strategies:

1. A comprehensive evaluation of the proposed approach is to be carried out by using additional datasets such as CIFAR-10 and ImageNet200.

2. In an effort to improve efficiency and scalability, machine learning applications will be divided into smaller, more manageable serverless functions.

3. Investigating resource utilisation and system scalability will be essential to ensure the successful operation of machine learning systems in edge environment.

1.3 Structure

In Section 2, the research reviews serverless, edge, and fog computing literature and compares pertinent studies. Section 3 explains the serverless edge deployment of the ML model using flowcharts and other technical graphics. Section 4 shows how the deployment generates a complete sequence of serverless edge platforms. Section 5 discusses the architecture, serverless functions, infrastructure tools, and output. Section 6 analyses each serverless function scenario in detail, providing its results. Section 7 outlines the study's findings and suggests further research to illustrate its worth to the cloud computing community in a serverless edge environment.

2 Related Work

Exploring ways to enhance and optimise serverless edge computing for machine learning tasks is crucial for computing field. The increasing implementation of serverless and edge computing technologies has generated curiosity over their utilisation for efficient data management and application execution in distributed environment. In-depth review of the literature, as well as the influence of serverless computing on enhancing machine learning tasks in distributed edge systems has been presented. It clarifies significant findings and identifies areas that are worthy of additional investigation. Every reviewed paper is carefully analysed to assess its contributions and limitations in supporting this research.

2.1 Integration of Serverless, Edge, and Fog Computing in Cloud Environments

Modern computer designs are reliant on serverless, edge, and fog computing especially in the field of cloud computing. Because serverless computing breaks down the computing unit into separate functions, it makes it possible to manage things more precisely and easily. This is crucial in cloud computing because it allows developers to focus on the code instead of the infrastructure by means of this abstraction, which guarantees both scalability and performance Nastic et al. (2022). In contrast, edge computing enables real-time processing and data analysis at the network's boundary, which is useful for IoT and cloud offloading applications. When it comes to optimising tasks across the Edge-Cloud continuum, the study conducted by Shi et al. (2016) offers valuable insights that highlight the restrictions imposed by resources and networks. Fog computing research from Bonomi et al. (2012) promotes the development of cloud computing to the edge of networks in order to facilitate the creation of innovative applications and services. However, the cost and scalability aspects of fog computing are still not well investigated, particularly in regards to the growing Internet of Things environments.

2.2 Serverless Computing Frameworks

Serverless frameworks offer a chance for research to be conducted, especially at the edge. To build and deploy serverless applications, these frameworks are essential. A serverless

framework called "Cirrus," created by Carreira et al. (2019), enables end-to-end machine learning workloads. In order to ensure effective machine learning operations in edge scenarios, this research takes into account data distribution, task scheduling, and resource limitations. It is significant to the proposed study since it emphasises the possible advantages of serverless frameworks for ML applications. However, understanding the Cirrus framework's mechanism suggests that it may not accurately reflect all edge environment situations in the real world. This may be a framework fault. It is necessary to overcome the developmental obstacles of serverless edge computing systems and their frameworks in order to implement the distributed paradigm of edge computing. These challenges include rapid evolution and limited resources. Palade et al. (2019) performed an in-depth evaluation of open-source serverless technological frameworks, and the results indicated that these frameworks offer advantages such as vendor neutrality and cost savings. The functional comparability of platforms such as Knative, OpenFaaS, and Apache OpenWhisk is confirmed by comparative analysis done by Li et al. (2021). However, there are differences in performance and scalability between these platforms. The study reveals that each of the four platforms has a comparable range of functionalities, including support for various programming languages, event-driven execution, and auto-scaling capabilities. Nevertheless, there exist differences in terms of performance and scalability. Kubeless and Apache OpenWhisk provide superior performance characteristics, whereas OpenFaaS and Knative demonstrate enhanced scalability capabilities.

2.3 Optimization and Cost of Serverless Applications

Since prominent cloud providers like AWS Lambda, Microsoft Azure Functions, and Google Cloud Functions have released their FaaS platforms, serverless computing has become an interesting technique and field of study. Several studies have highlighted the issue of the lack of performance and cost models, which results in uncertain performance and cost. Based on this information, a heuristic technique known as Probability Refined Critical Path Greedy (PRCP) uses four greedy algorithms to optimise the performance and cost of serverless apps Lin and Khazaei (2021). On average, the created model can estimate the cost and performance of serverless apps with a 98% accuracy rate, while the PRCP approach has an average accuracy of 97% in achieving optimal configurations, but it fails to provide a potential impact of factors such as network latency, data transfer costs, or variations in workload patterns on the performance and cost predictions. Similarly, a study conducted by Arora et al. (2021) from Deloitte showed that serverless deployment is growing, and over 75% of organisations polled have deployed or plan to use a serverless strategy within two years. Within the framework of AWS Lambda and EC2 instances, the research investigated server-based and serverless technologies and found that AWS Lambda saved 38% to 57% compared to a server-based cloud execution strategy. Which means significant cost savings over typical server-based approaches.

2.3.1 Performance Evaluation of Serverless Computing Platforms

As the cloud continues to grow, more and more server hosting choices have become available to businesses. These include serverless architectures, virtual machines, Docker containers. It is necessary to evaluate performance metrics for every service in order to make knowledgeable hosting selections Jain et al. (2020). Kumar and Selvakumara (2022) conducted a research which examines serverless architectures, including AWS Lambda

and IBM Open Whisk, and draws attention to their advantages, including scalability and cost effectiveness. There has been a movement towards open-source alternatives, such as Kubeless and knative due to worries about vendor lock-in caused by the widespread use of serverless solutions Kaviani et al. (2019).

A study published at IBM from Schweigert and Hadas (2022) has shown that cold starts can take anywhere from ten to fifteen seconds to more than a minute, which can have a devastating effect on serverless performance. This is in contrast to the efforts made to reduce cold start times in tools like IBM Cloud Code Engine, Red Hat OpenShift Serverless, and Knative. Especially for AWS Lambda, a popular option within the AWS suite used by many businesses, this presents a significant challenge Baird et al. (2017). According to Bardsley et al. (2018), in order to reduce serverless application latency, optimisation measures such as function warming and understanding of the fundamental architecture should be utilised. Performance testing and infrastructure management are highlighted in the work of Mahmoudi and Khazaei (2023), which uses models from queuing theory to understand the dynamics of serverless systems. To fully understand the complexities of serverless computing, this analysis can be better understood with the help of these serverless computing insights.

2.3.2 Serverless Computing Analysis for Edge-Based Machine Learning

To execute ML applications at the edge, serverless computing's ability to manage various workloads and reduce edge-based inference latency is becoming more critical. Kurz (2021) explores serverless architecture for distributed ML, enhancing knowledge of ML architectures and distributed learning approaches. Although not focused on edge computing, Christidis et al. (2019) offers new insights into deploying serverless architecture in resource-intensive scenarios. Furthermore, Trieu et al. (2022) illuminated the challenges of deploying machine learning applications at the edge and suggested that serverless computing may be a solution. Research shows that Kubeless responds faster to basic workloads than Fission, which struggles with high user demands. Research has shown empirical performance evaluations using tools like JMeter[1], assessing system performance across various workloads.

A study by Ishakian et al. (2018) highlights the significance of rigorous performance evaluations for deep learning models on serverless platforms. This study on MxNet-based deep learning raises AWS Lambda cold start and latency concerns. Cold starts may affect distribution delays, disrupting Service Level Agreements (SLAs).

In order to provide services that are both cost-effective and consistent, Cox et al. (2020) research on serverless inferencing in Kubernetes guides the implementation of efficient autoscaling and 'scale to zero' solutions. It presents the KFServing project, which is an extension of the KNative framework, and demonstrates how the benefits of serverless computing may be utilised for machine learning deployments. This demonstrates the need for developing solutions that improve the efficiency of machine learning tasks and the scalability of distributed edge systems.

In their investigation of the deployment of serverless edge computing for machine learning applications, Bac et al. (2022) make a significant contribution to this field by employing the MNIST dataset for training and testing in a distributed environment. The study acknowledges the demand for new optimisation measures and advocates for future research to validate these metrics. This is happening despite the fact that it has been

[1] `https://jmeter.apache.org/`

demonstrated that serverless computing may have potential benefits for machine learning applications. By carrying this research forward, the aim of the current research is to verify the additional metrics for optimisation in an edge environment by integrating existing findings. This allowed the research to address the weaknesses that were discovered in prior studies and provide a pathway with improved knowledge of the deployment efficiency of machine learning applications in serverless architectures.

2.4 Research Niche

The literature review identified several important gaps and obstacles in current studies on serverless edge computing, machine learning tasks, and improving performance in distributed edge systems. This work focuses on improving ML task efficiency in serverless, distributed edge computing. One way to accomplish this is that machine learning functions can be divided into smaller serverless functions to improve system manageability. Another way is to use additional metrics like the CIFAR-10 and ImageNet200 datasets to demonstrate the system's ability to perform under varied workloads. This research seeks to fill the gaps and overcome the limitations of previous research by investigating the use of specified datasets and the potential advantages of dividing machine learning systems into smaller, more manageable serverless functions.

- **Expected Contribution**

    1. **Incorporation of Additional Datasets:** The addition of datasets like CIFAR-10 and ImageNet200 to the proposed study will enhance machine learning workload evaluation in distributed serverless edge systems. Datasets like this allow the research to test how well and how versatile serverless edge computing is for machine learning.
    2. **Performance Evaluation with CIFAR-10 and ImageNet200:** This study replicated earlier evaluations of the serverless edge computing method by testing it on the CIFAR-10, ImageNet200, and MNIST datasets. Thoughtful decision-making is required to implement ML applications in edge environments, and this study clarified the approach's ability to manage complex and large-scale workloads.
    3. **Optimisation Strategy:** Simplifying ML apps' components assists serverless edge computing with machine learning workloads. This solution addresses serverless architecture startup and resource utilisation issues to improve distributed edge system efficiency and effectiveness.
    4. **Scalability Insights:** This research has used performance analysis with large datasets like ImageNet200 to investigate scalability metrics, specifically examining how the system handles increased loads and concurrent requests.
    5. **Identifying Resource Utilisation Across Nodes:** The project aims to validate key performance characteristics for machine learning workloads in serverless edge computing platforms. By following these steps, the methodology's benefits and drawbacks can be better understood, resulting in the effective implementation of machine learning systems based on the edge.

3 Methodology

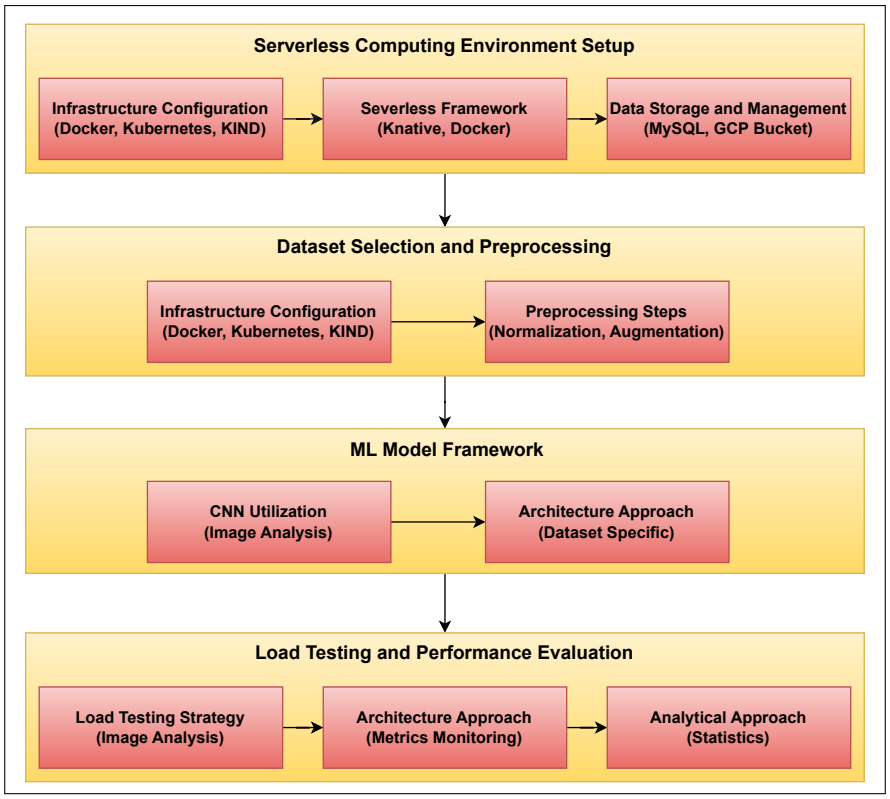


Figure 1: Workflow Diagram of Serverless Edge Computing System

This research's methodology is a quantitative experimental strategy for testing serverless edge computing framework effectiveness on ML workloads, and the workflow for the system is illustrated in figure 1. This research aims to optimise machine learning applications by breaking them down into smaller serverless functions and by integrating additional datasets as opposed to single datasets from the present studies available.

3.1 Serverless Computing Environment Setup:

- **Infrastructure Configuration:** By using Docker for Desktop, a Kubernetes cluster with three worker nodes and one master node is set up using KIND. A master node coordinates the launch and administration of serverless functions as the control plane, while a set of three worker nodes are set up to mimic an edge computing scenario.
- **Serverless Framework:** To enable autoscaling and event-driven execution, serverless functions are containerized using Docker and managed by Knative. The various components of the ML workload are handled by deploying functions that are tailored to processing and training datasets, managing models, and making predictions.
- **Data Storage and Management:** To store all of the model's metadata and serving as the central repository, the master node has a MySQL database installed. The use of Google Cloud Buckets for storing trained models provides scalable and secure data access.

3.2 Dataset Selection and Preprocessing:

- **Dataset Overview:** Three separate datasets—MNIST, CIFAR-10, and ImageNet200—are used in the study. To provide real-world image data, each dataset is picked for its unique properties and level of complexity.
- **Preprocessing Steps:** Datasets go through preprocessing steps like normalisation and augmentation to make them more robust and variable before they are fed into machine learning models.

3.3 Machine Learning Model Framework:

- **CNN Utilization:** The main machine learning model is convolutional neural networks (CNNs) because of how well they perform tasks involving images. When it comes to image datasets, these models perform well at extracting features.
- **General Architecture Approach:** Because each dataset is unique, the CNNs' architectures have been fine-tuned to adapt to the data available on MNIST, CIFAR-10, and ImageNet200, respectively, based on the pixel and image colour type, i.e., coloured or grey scale.
- **Data Storage and Management:** To store all of the model's metadata and serve as the central repository, the master node has a MySQL database installed. The use of Google Cloud Buckets for storing trained models provides scalable and secure data access.

3.4 Load Testing and Performance Evaluation:

- **Load Testing Strategy:** The k6 tool is used to load test the serverless functions. This program helps to understand the system's scalability and resilience by simulating different traffic patterns and workloads through a test script and Job execution.
- **General Architecture Approach:** To measure how well the system performs and how well it handles ML workloads, important metrics are monitored, including response time along with cold and warm starts, scalability, and resource utilisation.
- **Analytical Approach:** For understanding the results of the load testing, the research makes use of statistical techniques which includes histograms and charts. The purpose of this analysis is to derive useful findings on the effectiveness of the serverless architecture in an edge computing setting.

4 Design Specification

Advanced techniques and development methods are used to design this serverless edge computing system that performs well for machine learning. These phases determine how serverless functions preprocess, train, and predict from huge datasets like MNIST, CIFAR-10, and ImageNet200. This complex activity orchestration in a serverless environment is shown through a sequence diagram2. This section also provides a description of

the system's performance measures. It shows how well and efficiently the system operates in different operational conditions.

4.1 Serverless Function Roles and Interactions

The main parts of the system are the specialised Python serverless functions that are created to do specific jobs in the machine learning workflow:

- **preprocess_and_train:** Tasked with data preprocessing and training machine learning models for the MNIST, CIFAR-10, and ImageNet200 datasets. This function is tailored for each dataset, independently named as preprocess_and_train_mnist, preprocess_and_train_cifar10 and preprocess_and_train_imagenet, taking into account the particular needs of the datasets to provide optimised training performance.
- **predict:** It is designed to process the prediction requests received by connecting with trained machine learning models stored in the Google Cloud Bucket.
- **model_manager:** Oversees the Create, Read, Update, and Delete activities for the metadata of a model in the MySQL database.

4.2 Workflow and Data Flow

By sending a data processing request to the relevant preprocess_and_train function, the user or system starts the workflow. The Google Cloud Bucket stores processed data and trained models, whereas the MySQL database stores model information and references. The model_manager function can be requested to make it easier to manage models, while the predict function is used to retrieve and use the trained models to make predictions based on the model ID and image provided for prediction.

4.3 Sequence Diagram

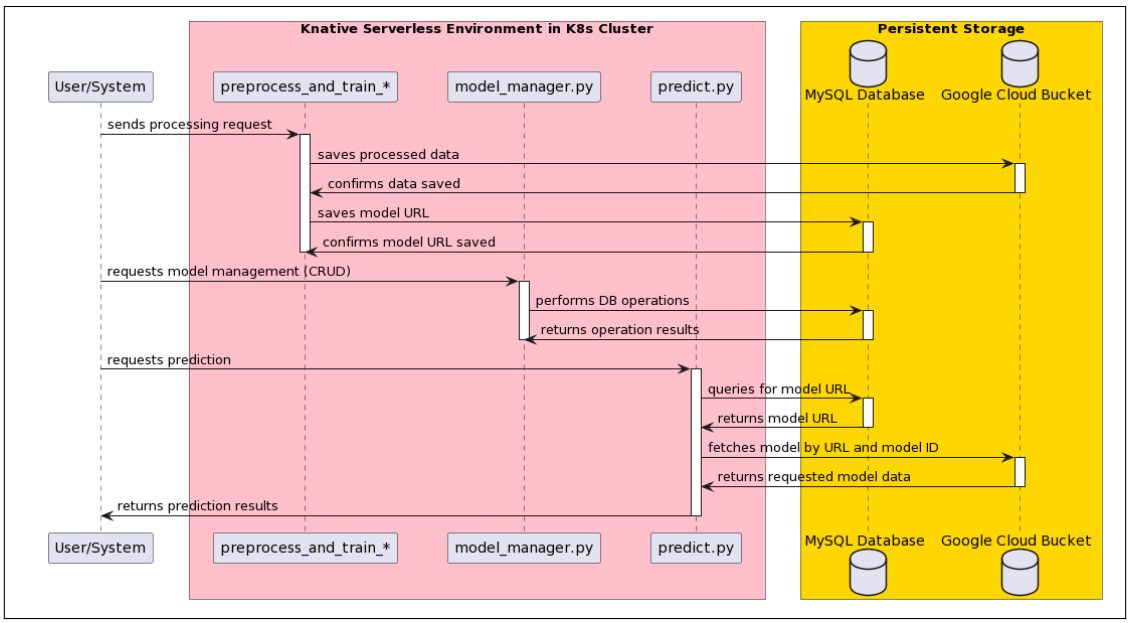


Figure 2: Serverless ML Model Sequence Diagram

The Figure 2 is representing the sequence diagram of the system's interactive flows and operational state. Its purpose is to show how data and actions move from the point when the user or system initiates a process all the way to the point where a prediction is made. It displays the system's and user's interactions with serverless services, the data transfer between the MySQL database and Google Cloud Bucket, and the system's overall response to different requests.

- **User/System Interaction:** A user or system-initiated action invokes the serverless functions to handle data, as shown in the Figure 2. This is where the process flow begins and where all the subsequent operations are initiated.
- **Data Preprocessing and Training:** Invoking the preprocess_and_train_* function prepares the dataset for training the model on CNN. These processes are essential for cleaning up raw data and making it suitable for machine learning model training.
- **Model Storage and Management:** When the training is complete, the model artefacts and metadata are safely saved in a Google Cloud Bucket. The MySQL database is used to record references. All of these processes for CRUD operations on the trained model are monitored by the model_manager function, which makes sure that the models are stored and can be retrieved for changes or modifications in the future.
- **Prediction Execution:** The predict function searches the MySQL database for the location and URL of the appropriate model based on the model ID provided whenever a prediction request is made. When the URL for the model is retrieved, it retrieves the model itself from the Google Cloud Bucket. This is the most important part of the process since it uses the trained model to make predictions using new data.

4.4 Performance Metrics

To evaluate serverless edge computing for machine learning workloads, a wide range of metrics have been used, along with a comparison with the findings found from the previous research studies Mahmoudi and Khazaei (2023); Nestorov et al. (2021). A tabular format similar to Table 1 has been used to systematically catalogue the findings of different operating situations.

Load/Virtual User	Avg Cold Start Duration	Avg Warm Start Duration	Total CPU Resource Utilization	Worker 1 Utilization	Worker 2 Utilization	Worker 3 Utilization
5-500	"To Be Measured" ms	"To Be Measured" ms	"To Be Measured"%	"To Be Measured"%	"To Be Measured"%	"To Be Measured"%
13-1100	"To Be Measured" ms	"To Be Measured" ms	"To Be Measured"%	"To Be Measured"%	"To Be Measured"%	"To Be Measured"%
20-1800	"To Be Measured" ms	"To Be Measured" ms	"To Be Measured"%	"To Be Measured"%	"To Be Measured"%	"To Be Measured"%
2-2100	"To Be Measured" ms	"To Be Measured" ms	"To Be Measured"%	"To Be Measured"%	"To Be Measured"%	"To Be Measured"%

Table 1: Template for Performance Evaluation of Serverless Functions under Varied Loads

Metrics Captured:

- **Load/ Virtual Users:** A specified number of virtual users are created using the K6 load testing scripts to stimulate requests to evaluate the performance of the designed system.
- **Average Cold Start Duration:** That average amount of time required to initialise serverless services when they are called for the first time upon deployment or when they have been idle were captured for the number of load tests performed.

- **Average Warm Start Duration:**For each load test, a average record how long it takes for serverless functions that have been initialised to respond when called later on is also counted.
- **Total CPU Resource Utilization:** The peak consumption was recorded for serverless functions during execution based on the load tests.
- **Worker Nodes Utilisation:** For each load test, the peak usage across all worker nodes was captured.

5 Implementation

An thoroughly planned architecture is brought to completion in this section, which covers the application's high-level architecture but also details the development of serverless edge computing system designed to enhance the efficiency of machine learning applications. Finally, this part digs into the implementation's last stage, highlighting the data outputs, ML model development, and tool and language suite used to get there.

5.1 System Architecture

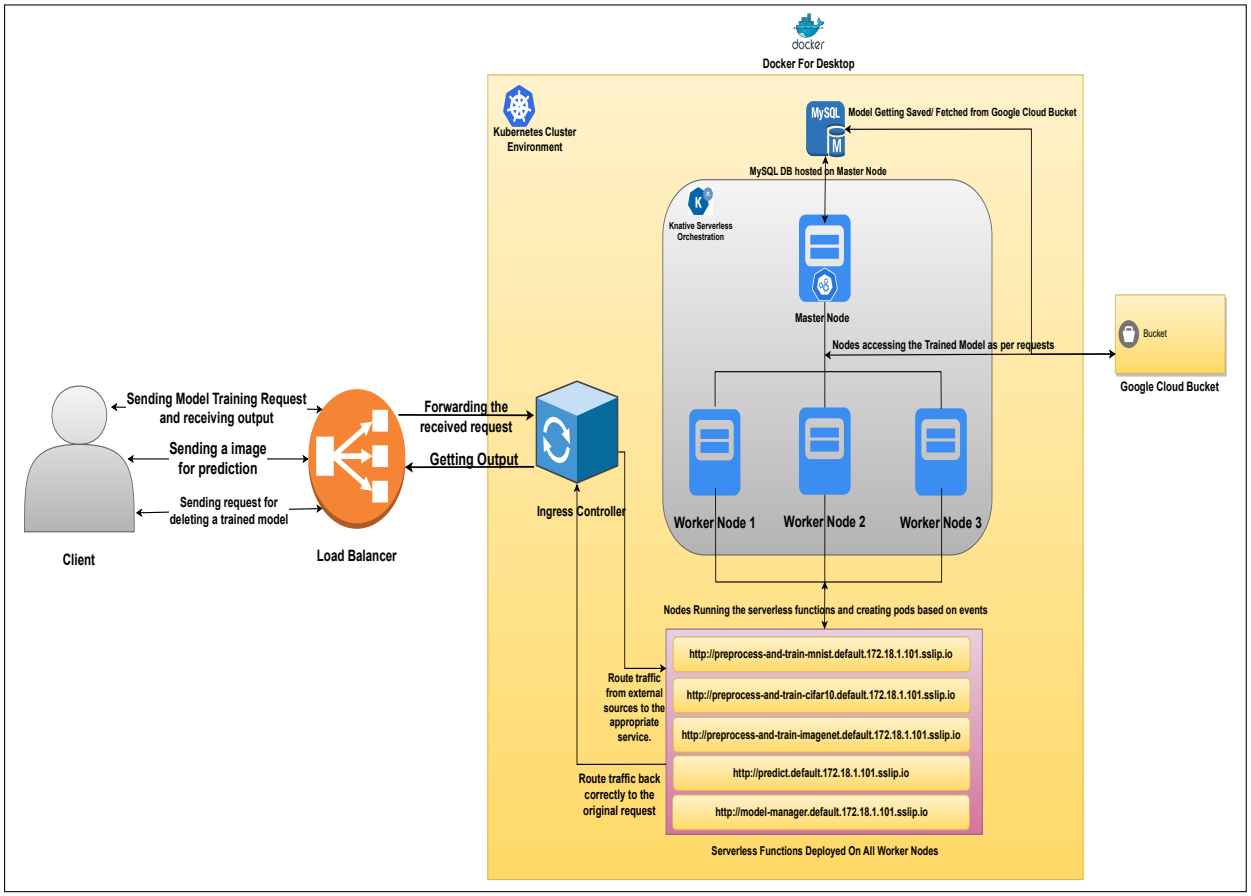


Figure 3: Serverless ML Model Architecture Diagram

The architecture in Figure 3 shows interaction between clients, serverless functions deployed on all three worker node and the services serving on the knative orchestration

along with the primary database storage solutions MySQL available in the Master Node. An efficient way for clients, such as smart devices and IoT sensors, to connect to the system is through a Load Balancer, which distributes requests throughout the Kubernetes cluster with Ingress controller in place.

The cluster is managed by Docker for Desktop and includes a Master Node that hosts a MySQL database and three Worker Nodes that execute serverless functions for machine learning activities. With containerization and deployment across all Worker Nodes, the serverless functions that handle data preprocessing-training, model updates, and prediction can be executed reliably. Machine learning models are stored in Google Cloud Bucket, from which Worker Nodes retrieve models as required for prediction jobs. The design makes use of k6 for load testing, logs to InfluxDB, and the Kubernetes Dashboard to keep tabs on how resources are being used and how the system handles traffic.

- **Outputs Produced:** Preprocessed and trained ML models, prediction based on requests received and model CRUD operations and a set of performance indicators that reveal how the system is running are the ends result and output resulted from this system. The data from the selected datasets which are MNIST, CIFAR-10 and ImageNet200 undergoes transformations to make it suited to ML processes, and models are built to be logically efficient.

5.2 System Development Tools & Infrastructure

The serverless edge computing system is built using several tools to develop and implement the machine learning models and Python as primary programming language.

1. **Programming Languages and Frameworks**
   (a) **Python:** Functions for serverless computing, data preparation, training of machine learning models, and prediction were build in python language.
   (b) **Flask:** It was chosen for developing the API endpoints for model training and prediction, and it was also ideal for model management using web services due to its lightweight nature.
2. **Code Management & Development Suite**
   (a) **Visual Studio Code (VS Code):** VS Code was the major coding tool. Its vast feature set, including Python and Docker extensions and GitHub version control, made it crucial to development. Code was continuously committed and published to GitHub from Visual Studio Code by following version control best practices and documenting code changes.
3. **Machine Learning Frameworks**
   (a) **TensorFlow and Keras:** All the resources needed to build, train, and validate CNN models were provided by these frameworks. TensorFlow was great for processing on the back end, and Keras was great for building neural networks because of its user-friendly interface.
4. **Convolutional Neural Network (CNN) Models**

(a) **MNIST Dataset:** A Keras model was constructed using a sequential architecture with MNIST, including of two convolutional layers, which were subsequently followed by max-pooling and dropout layers to mitigate overfitting. The model was completed with fully connected layers for the purpose of categorization.

(b) **CIFAR-10 Dataset:** The CIFAR-10 model used directly from the Keras framework has additional convolutional layers to effectively capture the complex patterns present in the dataset. Each convolutional layer was subsequently followed by max-pooling and dropout layers. The model concluded with densely linked layers that facilitated the classification process.

(c) **ImageNet200 Dataset:** Is has been collected from the official ImageNet website[2] and then the ResNet50 architecture was used as the foundational model, and subsequently modified with global average pooling and extensive layers to suit the specific needs of the ImageNet200 dataset.

5. **Containerization and Serverless Orchestration**

(a) **Docker for Desktop:** The serverless functions were capable of being containerized, guaranteeing that the environments used for development and production were identical.

(b) **Kubernetes:** Creating a Kubernetes multi-node cluster with help of KIND with the orchestration of containerized apps, overseeing their deployment, scaling, and cluster operation was achieved with this setup.

(c) **Knative:** An important resource for Kubernetes environments that offered serverless orchestration capabilities. Knative managed the lifecycle of serverless workloads, simplified the deployment of containerized apps, and allowed the system to dynamically scale the number of pods up or down depending on demand.

6. **Master Node Setup and Cloud Storage**

(a) **MySQL:** MySQL was installed as the database on the Master Node to store model metadata and other essential information.

(b) **Google Cloud Bucket:** Took advantage of GCS to store and retrieve machine learning models, ensuring scalability and effortless integration with the serverless architecture.

7. **Monitoring and Load Testing**

(a) **Kubernetes Dashboard:** It assisted for visualisation of resource utilisation inside the cluster and it made this available through the provision of its monitoring dashboards.

(b) **k6:** An effective tool for load testing that used specially designed scripts to stimulate traffic and allowed to analyse how the system worked in a range of load conditions.

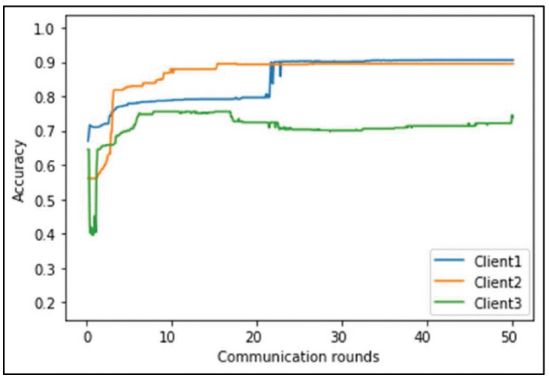
(c) **InfluxDB:** Through its less resource intensive approach InfluxDB assisted to do further analysis as the performance data was captured and kept while the load tests scripts were executed with the k6 job.

[2] `https://image-net.org/index.php`

6 Evaluation

To evaluate the serverless computing architecture that was built as a result of this research, a thorough examination of the system's performance in handling machine learning workloads was necessary. This section highlights the findings that provide evidence to support the research question and objectives. To thoroughly analyse the experimental data and determine the significance of those results, statistical approaches are used.

6.1 Model Accuracy Across Nodes



MNIST Old Model vs. MNIST, CIFAR-10 and ImageNet200 New Model Accuracy

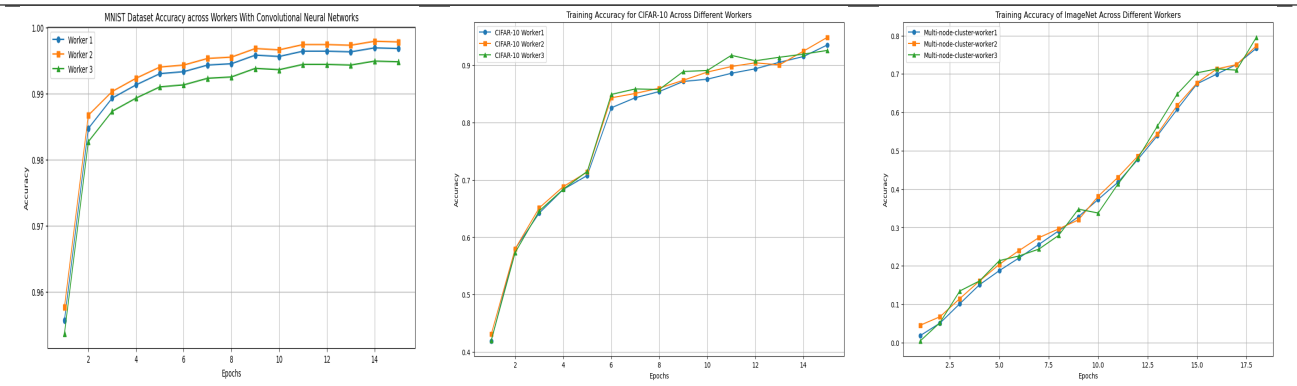


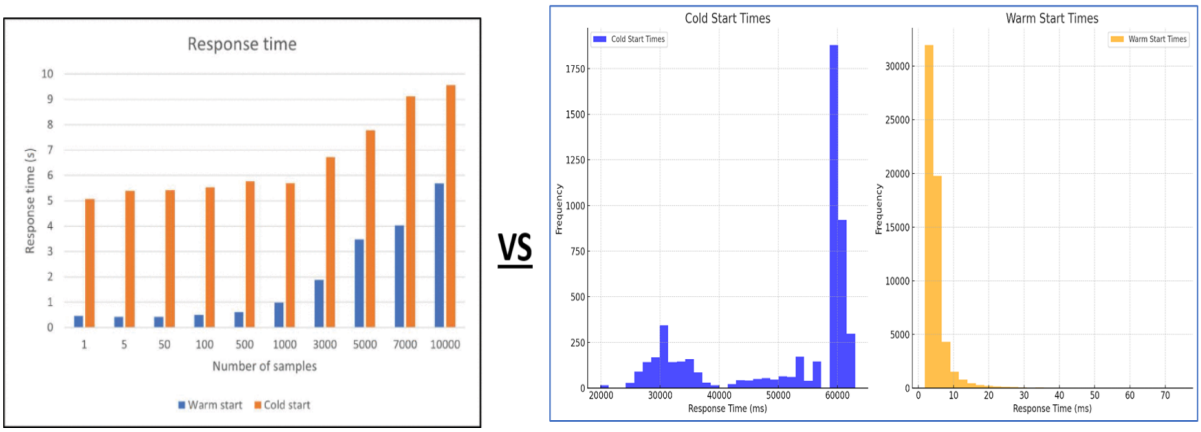
Table 2: Model Training Accuracy Comparison of Old Vs New System

Training accuracy on the MNIST, CIFAR-10, and ImageNet200 datasets was evaluated across all worker nodes in the current system for various number of epoch which is seen in the table 2 which compares the accuracy. Training accuracy on the MNIST dataset was consistent, and models were successfully saved and retrieved using Google Cloud Bucket. The models demonstrated strong performance across several nodes in the CIFAR-10 and ImageNet200 datasets, following similar performance of MNIST.

The new system highlights significantly higher training accuracy and demonstrates a sustained performance across all nodes when compared to the old system which is observed in the table 2 demonstrating varying performance levels on each node which is evidence of how the optimised serverless environment improved performance. It is clear from the better accuracy percentages found across all three nodes that the new architecture allowed for more efficient data handling and model training procedures.

6.2 Cold Start vs. Warm Start Performance

Critical to serverless computing, response times during cold and warm starts of both old and the new proposed system are evaluated as part of the system's evaluation.



Response time of Old System when trigger serving function

Response time of New System when trigger serving function

Figure 4: Response time comparison of the old system (left) versus the new system (right) for cold and warm starts.

- **Comparative Response Time Analysis:** Cold start and warm start entries were represented in the histograms seen on the right side of the Figure 4, which allowed for visualisation of the analysis. Most warm starts were conducted in less than 10 milliseconds, and the new system displayed 5,201 cold start entries and 59,956 warm start entries, indicating a higher frequency and efficiency in responding to subsequent requests. When compared to the old system, the new one performed far better at cold start latency management and reduction while maintaining warm start processing times for huge number of requests.

6.3 Load Testing and Scalability

A series of load tests was carried out with a range of virtual users ranging from 1 to 2,100. This was done to evaluate system's scalability. It was clear from the results that the system could keep working at a high level of accuracy and speed until users count reached 1875. Despite the fact that a small number of requests resulted as interrupted, the system was able to maintain an accuracy level that was close to 99% under a variety of load conditions.

- **Scalability Test Results**

1. **Testing Performance with Users scaling from 5-500:**

   As seen from the Figure 5, load tests with 5-500 users was executed to determine how much of the system's resources were used. The system is not overloaded because the CPU use peaks does not exceed the available capacity with stable memory use indicates that the workload is well within the system's capabilities. There were no interruptions in the processing of any of the 287,092 requests that were successfully processed from the load test performed through k6. Request handling times and data transfer rates are stable, which shows that the system is fast. Overall, the

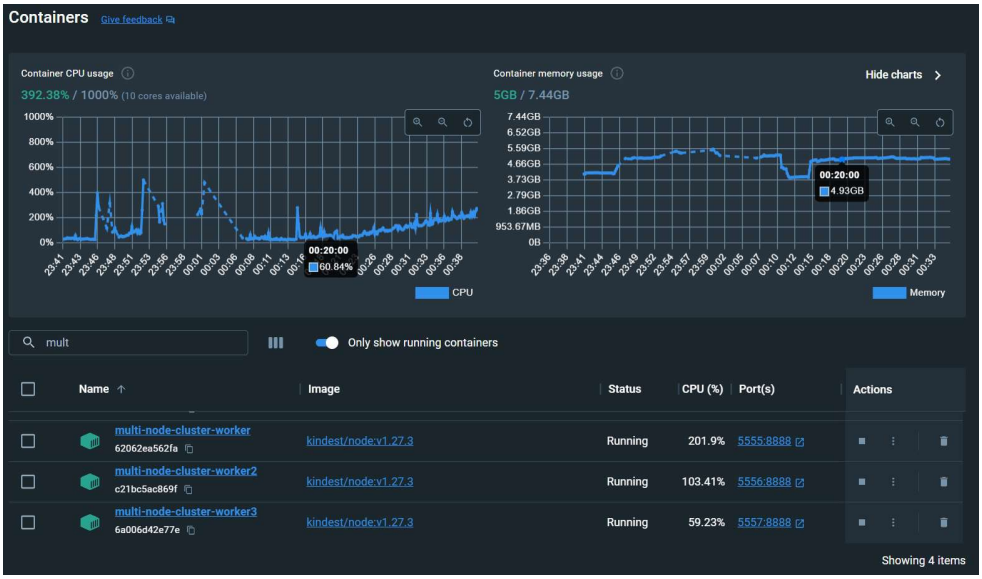


Figure 5: Resource utilization during load test for 5-500 Virtual Users

testing with users scaling till 500 shows that the system works well and can grow with the number of users that were tried. It does a good job of managing resources and keeping service quality high.

2. **Testing Performance with Users scaling from 13-1100:**

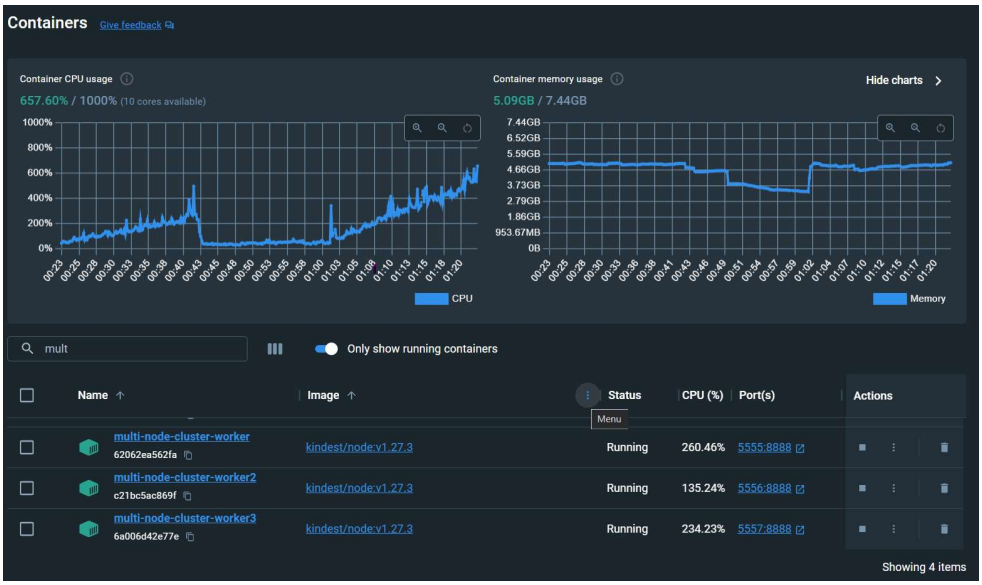


Figure 6: Resource utilization during load test for 13-1100 virtual users.

In this test case, 13-1100 virtual users raise CPU use significantly. The graph in Figure 6 indicates about 657.60% usage of the 10 cores, indicating a significant workload. The system appears to have memory capacity to handle increased traffic, as worker nodes use memory according to request processing and pods scale on each node. Memory utilisation remains consistent and stable at 5.09GB out of 7.44GB.

From 13–1100 virtual users, the system executed all 787,052 requests without error. The system managed network connections efficiently despite increased users

because HTTP request metrics including waiting, receiving, and connecting times were low. Iterations per second, virtual users, and maximum virtual users match the test configuration and load parameters. The system's strong performance and consistent quality of service even as the load increases showcases an efficient and well-configured environment.

3. **Testing Performance with Users scaling from 20-1800:**

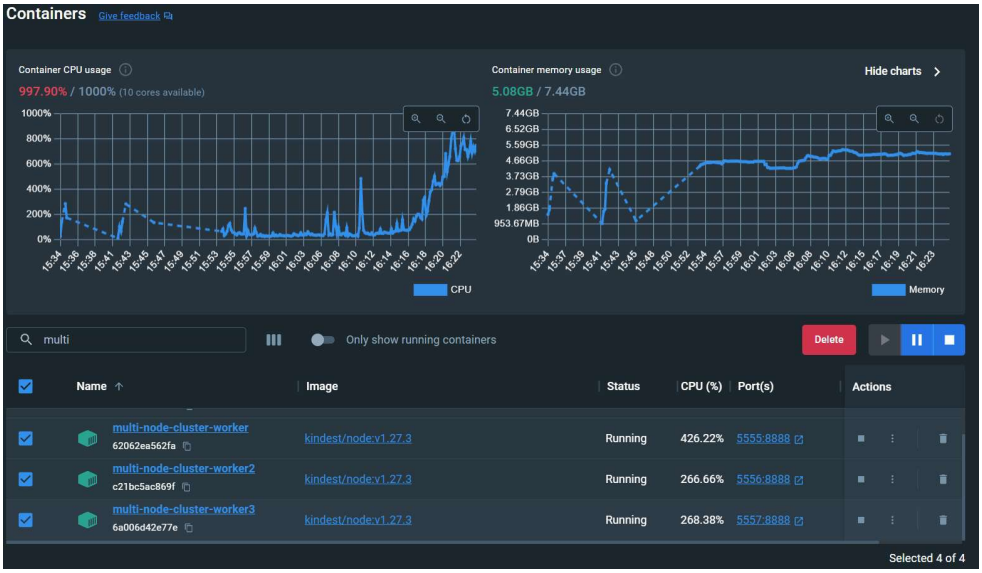


Figure 7: Resource utilization during load test for 20-1800 Virtual Users

Kubernetes Dashboard (Figure 7) shows the system's ability to manage multiple virtual users. The system effectively handled large demands, with CPU utilisation peaking at 997.90% (Figure 7) and worker node 1 and other nodes managing requests concurrently. The system demonstrates effective memory management through consistent memory usage, accompanied by a minor increase in system capacity. Stability and reliability were evident when the system processed 727,016 requests with 99.98% accuracy. Despite a consistent increase in virtual user numbers and high iteration counts, the system maintained its performance under rising demand. These numbers show that the system can handle large traffic and be efficient.

4. **Final Test Performance with Users scaling from 2-2100:**

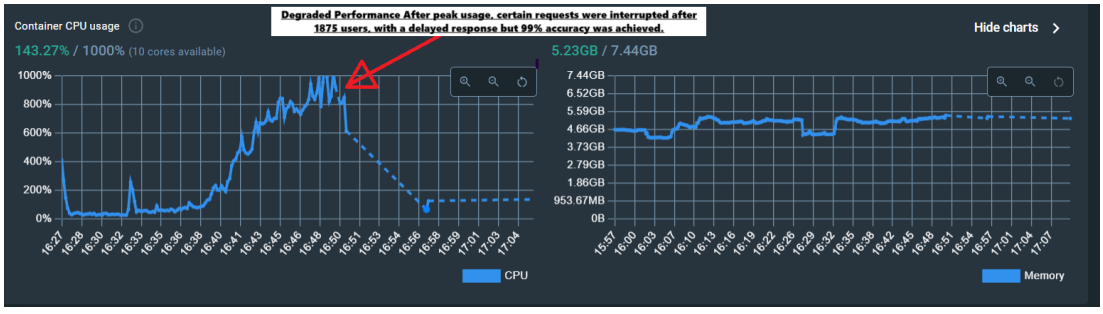


Figure 8: Dashboard showing performance degrade after reaching system peak

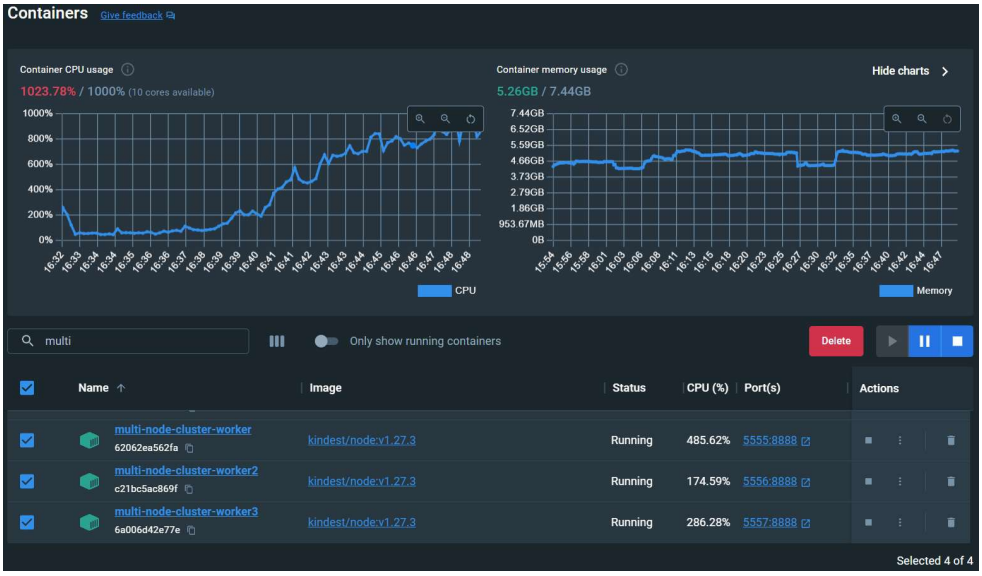


Figure 9: Resource utilization during load test for 2-2100 virtual users

The system showed remarkable robustness throughout the scalability test for 2-2100 virtual users, as shown in Figure 9. The system was strained beyond its usual operational boundaries as the CPU utilisation peaked at 1023.78%, surpassing the nominal 1000% level. Consistent memory utilisation in situations of high demand is indication of well-managed memory resources. A record 872,218 requests were fulfilled by the system throughout the test. Even when under heavy demand, the system's memory usage remained consistent at around 5.26 GB out of 7.44 GB.

However, the system encountered 1082 interrupted iterations when the user load exceeded 1875 users and system's performance started to decline as seem in Figure 8, indicating its maximum capability under great stress. Even though there was a degrade in performance the system was still able to achieve a 99% of successful request completion and a acceptable response time for 97% of requests. The test findings highlight the importance of optimising the application or scaling resources to accommodate these peak loads without service interruptions.

According to results from the load test, the system is capable of handling a huge amount of traffic, but beyond a certain point, performance starts to degrade as expected from the designed system, which illustrates how much peak the system can handle and survive efficiently.

6.4 Knative Autoscaling Evaluation

Under a simulated load of 2100 virtual users, the Knative autoscaling capabilities were tested, as shown in Figure 10. Over the course of 23 minutes, the autoscaling graph displays the quantity of pod instances. Knative dynamically expanded the number of pods to handle the incoming traffic, reaching a peak of 43 pod instances, from an initial minimal number when user load grew. Since the pod count changes with user requests, the autoscaling technique is operating as expected. But when the jobs gets completed after 23-minute, the pod count drops rapidly, signalling that user load has dropped and there's no longer any need for a large number of pods. So, it analyses the situation and decides to scale down the pods until they reach zero.

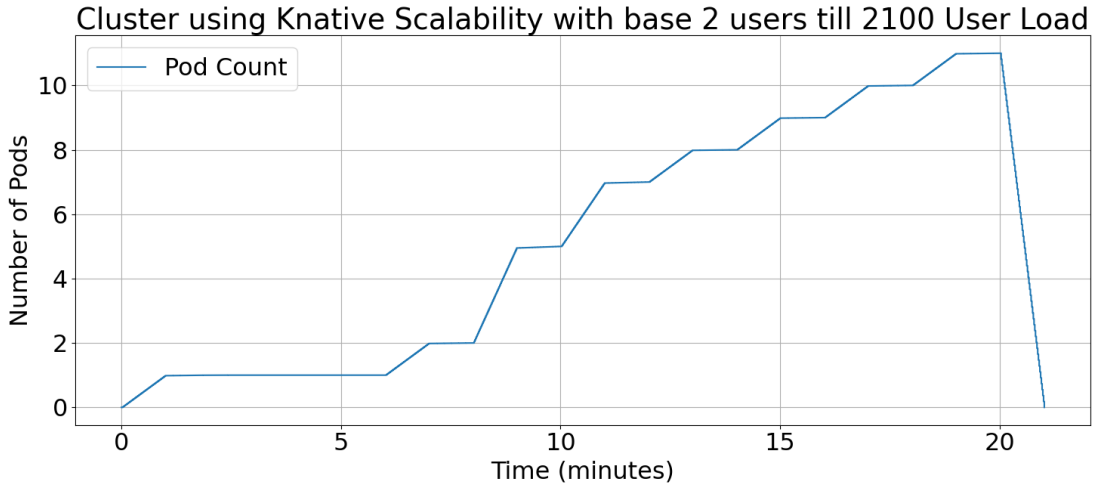


Figure 10: Knative Autoscaling for Load Management

As part of the load balancing process, instances are created and terminated dynamically in pods. While most pods are in a healthy "Running" condition, the existence of "Terminating" pods informs that there is ongoing scaling down activity following peak load management. The graph provides evidence of how Knative's serverless environment is elastic, which is important for keeping services available and performing well under different load situations.

6.5 Discussion

The experiments conducted provided a evaluation of the serverless architecture's performance under varied loads. Table 3. summarizes the results, presenting a view of how response times and CPU utilization was affected as the number of virtual users increased.

Load/Virtual User	Avg Cold Start Duration	Avg Warm Start Duration	Total CPU Resource Utilization	Worker 1 Utilization	Worker 2 Utilization	Worker 3 Utilization
5-500	23.5 s	3.22 ms	392.38%	201.9%	103.41%	59.23%
13-1100	24.7 s	2.61 ms	657.60%	260.46%	135.24%	234.23%
20-1800	24.4 s	4.9 ms	997.90%	426.22%	266.66%	268.38%
2-2100	39.98 s	12 ms	1023.78%	485.62%	174.59%	286.28%

Table 3: Performance Evaluation of Serverless Functions under Varied Loads

The data presented in Table 3 shows that cold start event length is directly proportional to user count. The first three rows shows the system is able to sustain the users with excellent performance and average cold start response less than 25 seconds. This finding supports the idea that under normal circumstances, a cold start can take up to 10-15 seconds initialization delay. In contrast, the warm starts constantly maintained a low level, indicating that once initialised, the instances were capable of handling new requests with minimal latency. The finding aligns with existing research, highlighting the need of implementing tactics that reduce cold start duration in a resource intensive and a workload where huge amount of requests needs to be served to enhance the user experience in serverless apps.

The evaluation showed that the system's auto-scaling feature with Knative can handle more simultaneous users while retaining optimal performance, surpassing the old system. Scaling is essential for managing unpredictable workloads and applications with changing traffic. Knative implementation optimised resource allocation across worker nodes. The

system optimised CPU and memory utilisation through dynamic resource allocation and deallocation, leading to improved warm start response in all tests (see to Table 3).

The CPU utilisation showed a nearly proportional rise in response to the workload, indicating the serverless platform's ability to automatically scale. However, in the highest demand period, as observed in the final row of Table 3 for 2100 users, the CPU usage surpassed the allocated capacity and then allocated extra resources as the cluster was configured to allocate a extra 25% in case of resource peak utilisation, but if was not configured for a extra 25% resource allocation during request burst events an over allocation that may result in increased costs during real world scenarios.

7 Conclusion and Future Work

The purpose of this research was to investigate serverless edge computing for distributed machine learning applications in great detail with the use of open-source technologies. With new datasets like CIFAR-10 and ImageNet200 and the splitting of ML applications into smaller serverless functions, performance and efficiency have improved. A number of performance tests proved the implementation was valid, showing that it improved response times during cold and warm starts, scaled well under different user loads, and made the most efficient use of resources on each node.

Results indicate that serverless designs enhance edge computing operations. In particular, the system showed good scalability, efficiently managing increased loads while maintaining high performance accuracy. Research has enhanced understanding of cold start times and demonstrated that optimisations can significantly reduce latency, resulting in an improved user experience. Additionally, Knative is crucial for system scalability, ensuring smooth handling of compute demands. Dynamic resource allocation and deallocation on serverless platforms optimise operating expenses by matching resource utilisation with demand, highlighting their efficiency.

An encouraging basis for improving performance and making optimal use of resources has been laid out by the present study into serverless edge computing for machine learning. Research in the future can build on these findings to investigate various strategic improvements, such as: The use of federated learning to improve the accuracy of models learned on massive datasets such as ImageNet should be thoroughly investigated. Splitting the dataset and training it on different nodes allows each node to train a particular portion of the dataset to create a more complete and accurate global model that can be created by combining the data the from nodes. This method helps with data analysis while still protecting individual privacy, and it also improves accuracy. The goal of parallelized model training is to create methods that can train huge models over multiple server nodes in parallel. Because of this, it might improve the system's ability to handle huge machine learning workloads and reduce training time. To maintain the accuracy and reliability of the node models in the combined model synthesis, it is important to plan for parallelization. Modern Methods for Optimisation: Hyperparameter tuning and neural architecture search (NAS) are two sophisticated optimisation methods that can be investigated to further optimise machine learning models. In a serverless edge environment, NAS can automate neural network architecture design to identify more efficient models than manually created models.

References

Arora, G., Tayal, A. and Sembhi, R. (2021). Determining the total cost of ownership: Comparing serverless and server-based technologies, Deloitte Consulting.
URL: *https://d1.awsstatic.com/SMB/deloitte-tco-of-serverless-whitepaper-2022-smb-build-websites-and-apps-resource.pdf*

Bac, T. P., Tran, M. N. and Kim, Y. (2022). Serverless computing approach for deploying machine learning applications in edge layer. Date Added to IEEE Xplore: 26 January 2022.

Baird, A., Huang, G., Munns, C. and Weinstein, O. (2017). Serverless architectures with aws lambda.
URL: *https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf*

Bardsley, D., Ryan, L. and Howard, J. (2018). Serverless performance and optimization strategies, *2018 IEEE International Conference on Smart Cloud (SmartCloud)*, IEEE, New York, NY, USA.

Bonomi, F., Milito, R., Zhu, J. and Addepalli, S. (2012). Fog computing and its role in the internet of things, *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, ACM, pp. 13–16.

Carreira, J., Fonseca, P., Tumanov, A., Zhang, A. and Katz, R. (2019). Cirrus: A serverless framework for end-to-end ml workflows, *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, ACM, pp. 13–24.
URL: *https://doi.org/10.1145/3357223.3362711*

Christidis, A., Davies, R. and Moschoyiannis, S. (2019). Serving machine learning workloads in resource constrained environments: a serverless deployment example, *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, IEEE.

Cox, C., Sun, D., Tarn, E., Singh, A., Kelkar, R. and Goodwin, D. (2020). Serverless inferencing on kubernetes, *Distributed, Parallel, and Cluster Computing* .
URL: *https://doi.org/10.48550/arXiv.2007.07366*

Ishakian, V., Muthusamy, V. and Slominski, A. (2018). Serving deep learning models in a serverless platform, *2018 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, Orlando, FL, USA.
URL: *https://ieeexplore.ieee.org/document/8360337*

Jain, P., Munjal, Y., Gera, J. and Gupta, P. D. (2020). Performance analysis of various server hosting techniques, *Procedia Computer Science* **173**: 70–77. International Conference on Smart Sustainable Intelligent Computing and Applications under ICITETM2020.

Kaviani, N., Kalinin, D. and Maximilien, M. (2019). Towards serverless as commodity: a case of knative, *WOSC '19: Proceedings of the 5th International Workshop on Serverless Computing*, IBM Publications, pp. 13–18.

Kumar, N. S. and Selvakumara, S. S. (2022). Serverless computing platforms performance and scalability implementation analysis, *2022 International Conference on Computer, Power and Communications (ICCPC)*, IEEE.

Kurz, M. S. (2021). Distributed double machine learning with a serverless architecture, *Companion of the ACM/SPEC International Conference on Performance Engineering*, ACM, pp. 27–33.

Li, J., Kulkarni, S. G., Ramakrishnan, K. K. and Li, D. (2021). Analyzing open-source serverless platforms: Characteristics and performance, *The 33rd International Conference on Software Engineering & Knowledge Engineering*. [Submitted on 4 Jun 2021].
URL: *https://arxiv.org/abs/2106.03601*

Lin, C. and Khazaei, H. (2021). Modeling and optimization of performance and cost of serverless applications, *IEEE Transactions on Parallel and Distributed Systems* **32**(3): 615–632.

Mahmoudi, N. and Khazaei, H. (2023). Performance modeling of metric-based serverless computing platforms, *IEEE Transactions on Cloud Computing* **11**(2): 1899–1910.

Nastic, S., Raith, P., Furutanpey, A., Pusztai, T. and Dustdar, S. (2022). A serverless computing fabric for edge & cloud, *2022 IEEE 4th International Conference on Cognitive Machine Intelligence (CogMI)*, IEEE, Atlanta, GA, USA.

Nestorov, A. M., Polo, J., Misale, C., Carrera, D. and Youssef, A. S. (2021). Performance evaluation of data-centric workloads in serverless environments, *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, IEEE, IEEE, Chicago, IL, USA.

Palade, A., Kazmi, A. and Clarke, S. (2019). An evaluation of open source serverless computing frameworks support at the edge, *2019 IEEE World Congress on Services (SERVICES)*, IEEE, Milan, Italy. Conference: 08-13 July 2019.

Schweigert, P. and Hadas, D. (2022). Reducing cold start times in knative, red hat openshift serverless, and ibm cloud code engine.
URL: *https://developer.ibm.com/articles/reducing-cold-start-times-in-knative/*

Shafiei, H., Khonsari, A. and Mousavi, P. (2022). Serverless Computing: A Survey of Opportunities, Challenges, and Applications, *ACM Computing Surveys* **54**(11s): 1–32. Article No.: 239.
URL: *https://doi.org/10.1145/3510611*

Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016). Edge computing: Vision and challenges, *IEEE Internet of Things Journal* **3**(5): 637–646.

Trieu, Q. L., Javadi, B., Basilakis, J. and Toosi, A. N. (2022). Performance evaluation of serverless edge computing for machine learning applications, *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, IEEE.