Fall 2023

Serverless Architecture for Machine Learning

Ikshaku Goswami

Serverless Architecture for Machine Learning

A Project Report

Presented to

Dr. Robert Chun

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Ikshaku Goswami

Dec 2023

The Designated Project Committee Approves the Project Titled

Serverless Architecture for Machine Learning

by

Ikshaku Goswami

Approved for the Department of Computer Science

San Jose State University

December 2023

Dr. Robert Chun	Department of Computer Science
Dr. Navrati Saxena	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science

Acknowledgements

I extend my sincere gratitude to Dr. Robert Chun, my esteemed project advisor, for his invaluable guidance, unwavering encouragement, and steadfast support throughout the entire duration of this project. His expertise has been instrumental in shaping the trajectory of my work.

I would also like to express my heartfelt thanks to Dr. Navrati Saxena and Dr. Thomas Austin, esteemed members of my committee, for graciously agreeing to be part of this academic journey. Their insightful guidance and constructive feedback have significantly enriched the quality of my research.

My deepest appreciation goes to my family; their unwavering support and understanding have been the bedrock of my perseverance. Without them, reaching this milestone would not have been possible.

Abstract

Serverless computing is an area under cloud computing which does not require individual management of cloud infrastructure and services. It is the groundwork behind Function as a Service or FaaS cloud computing technique. FaaS provides a stateless event-driven orchestration of functions and services for applications deployed in the cloud, without having to manage the servers and other infrastructure resources. This event driven architecture is being well utilized to manage different web-applications and services. Machine learning can bring a unique challenge to serverless computing, as it involves high-intensive tasks which requires voluminous data. In such a scenario it becomes essential to optimize the cloud-deployment architecture to obtain accurate results efficiently. In addition, serverless computing suffers from drawbacks like cold start etc., which further increases the need of researching different serverless provisioning tools and techniques. This research work aims to deploy a machine learning model to detect real-time crisis, using various serverless computing resources provided by notable cloud vendors like Amazon Web Services (AWS) and Google Cloud Platform (GCP). It also compares among the various methodologies available and later aims to build a training platform for machine learning tasks.

Index Terms – Serverless, AWS, Lambda, S3, EFS, EC2, VM, GCF

TABLE OF CONTENTS

List of Figures

List of Tables

1. Introduction

The evolution of Machine Learning and Artificial Intelligence have been a longstanding arduous process. It bears fruits in our technological milestones of today, wherein it is being heavily used to modernize our healthcare, business, education, and a lot of other sectors. Enabling computers to learn from patterns, Machine Learning has made it easy to make accurate and precise decisions. Today, different areas of Machine Learning are developing rapidly such as Natural Language Processing or NLP. NLP models are used to infer context from text, speech etc., and make decisions based on it. A good example of it is question answer prediction. Researchers have been able to build models like BERT, to accurately answer quizzes. They have been trained on a large corpus of articles, documents etc., from the internet and derive accurate answers based on the context of the question as well. Furthermore, AI models are bringing revolutionary changes when it comes to automation like self-driving cars etc.

Machine Learning has brought forward tremendous changes in our day-to-day lives, but it is still a highly resource intensive technology. To train models, it takes a huge amount of CPU and GPU resources which are not unlimited. Especially when it comes to cost of these resources, it can play a huge overhead for companies and individuals who want to experiment and test these models for their use cases. Especially industries today heavily rely on outsourcing these resources which brings us to the advent of cloud computing.

Cloud computing has made it easier for everyone to manage their resource requirements. Today, to host or run applications, databases etc., users can simply spin up servers in the cloud and utilize those. This has brought down the costs

associated with managing these infrastructure resources as most of the cost model is based on Pay-as-you-Go structure, whereby you pay only for the time the resources are being utilized for. Cloud providers like Amazon Web Services (AWS), Google Cloud Platform (GCP) or Azure have taken a great leap forward by also providing and managing resources for training and building machine learning models. These models can be highly resource and cost intensive, however, using distributed cloud networks and servers, users can bring those down drastically. Even starting from housing data in cloud rather than on-prem can add to the cost credits. Together with that, users are now running the models on cloud server like EC2 machines for AWS. Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a virtual machine (VM) or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. Today, users are using this technique to also deploy their machine learning models in the cloud.

Serverless is an important cloud architecture. It falls within the FaaS or Function as a Service architecture model. In this framework users are not privy to the underlying infrastructure for their applications. Many cloud providers like AWS, Google and Azure provide serverless resources. One of the most used serverless resource is AWS Lambda which will be discussed further in this paper. Containerization is heavily used during this process of serverless orchestration of resources and will be discussed further as well. When it comes to machine learning, nothing is more beneficial than serving the models in real-time. This is where the serverless nature

of cloud computing can bring a lot of benefits. As resources are spun up automatically without manual intervention, machine learning tasks can be divided and addressed in a distributed way which could lead to faster rendering of results.

1.1 Problem Statement

This research project deals with developing machine learning inference and training architectures using core serverless offerings from AWS and GCP. These architectures will then be evaluated and ranked based on cost and time to train or execute. Furthermore, research will also be undertaken to build a training model using core AWS serverless resources like Elastic Container Service (ECS), to address its feasibility for real-world solutions. While the bulk of the focus is on researching serverless frameworks, the project also deals with critical machine learning tasks for building a model that can predict whether a given input text could be related to a disaster or not. Different sized models will be built, which would then be ran as a serverless service for inference or trained in cloud using serverless framework like ECS.

1. 2 Motivation

Traditional machine learning techniques usually involve developing models and improving their results. As can be realized, these techniques can be complemented with serverless cloud computing. However, there are challenges involved in this framework. Particularly when it comes to constraints like data and memory limits of the serverless functions. Adding to this are other problems which will be discussed further like cold start of the functions. In view of these challenges and the benefit that can be drawn, the aim of this research project is to render real time models for crisis detection using serverless architecture from AWS and Google.

2. Background

Cloud Computing

Cloud computing as a technology is ubiquitous, with its growing presence felt as the adoption of internet becomes widespread. In essence, cloud computing leverages remote servers to provide resources like compute, infrastructure, software etc., using the internet. It unleashes scalability through its on-demand model and allows users to access and use any computing resource without having to invest or manage them extensively. Cloud computing has several key features, such as cost-effectiveness because users only pay for the resources they use, flexibility because users may scale resources up or down based on their needs, and accessibility because users can access data and apps from any location with an internet connection. There are a growing number of cloud providers like Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure etc. Typically cloud computing is categorized into three service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS)

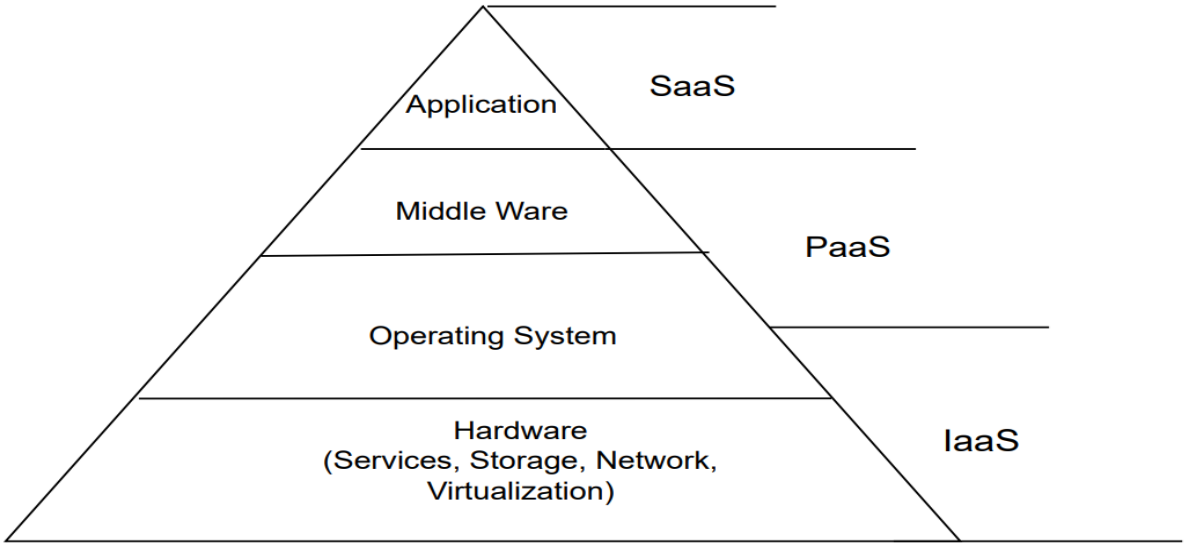

Fig 1 – Cloud Computing Service Model

Infrastructure as a Service (IaaS): IaaS utilizes the internet to provide virtualized computing resources such as storage, virtual machines, networking components etc. Users do not have to deal with the hassle of managing these components as they are managed by the cloud providers, which saves a significant portion of their costs. Example – AWS Elastic Compute (EC2) virtual instances.

Platform as a Service (PaaS): PaaS offers developers with a streamlined approach for creating applications and writing code by managing all the required underlying infrastructure orchestration. Example – Heroku, which is a cloud deployment tool, allows a straightforward deployment of code, scaling of infrastructure and database integration among others.

Software as a Service (SaaS): SaaS uses the internet for the distribution of software applications. These applications can be accessible to users through a web browser, that removes the requirement for local installation and streamlines updates and maintenance. Example – Microsoft 365 suite.

Function as a Service (FaaS)

Function as a Service (FaaS) or commonly referred to as Serverless Computing is another growing service model within cloud computing. It stems from IaaS and aims to further abstract out the orchestration and management of computing resources, specifically virtual machines like EC2 instances. For any developer, running EC2 instances, maintaining them, and scaling them was always a pain-point as it not only involved incurring costs but also wasting valuable developer time. To tackle this, cloud providers like AWS, Azure, GCP have introduced the concept of FaaS, which are on-demand event-driven execution environments that gets automatically provisioned to run code. These functions are stateless in nature, with very limited run-time memory and storage limits. The highest allotted runtime memory is 10GB provided by AWS Lambda, which is AWS's serverless offering. An example of serverless function usage could be an API call made to a lightweight service (say a service to send an automated email given a user sign up for a newsletter). In this case this microservice could be written as part of a serverless offering such as a Lambda function that gets automatically triggered when a user POST data to an API endpoint upon signing up for a newsletter. Below is a salient representation of the backend of an application built using serverless architecture. The noteworthy attribute is that the scaling of the function's is taken care of by the cloud provider itself depending on the traffic.

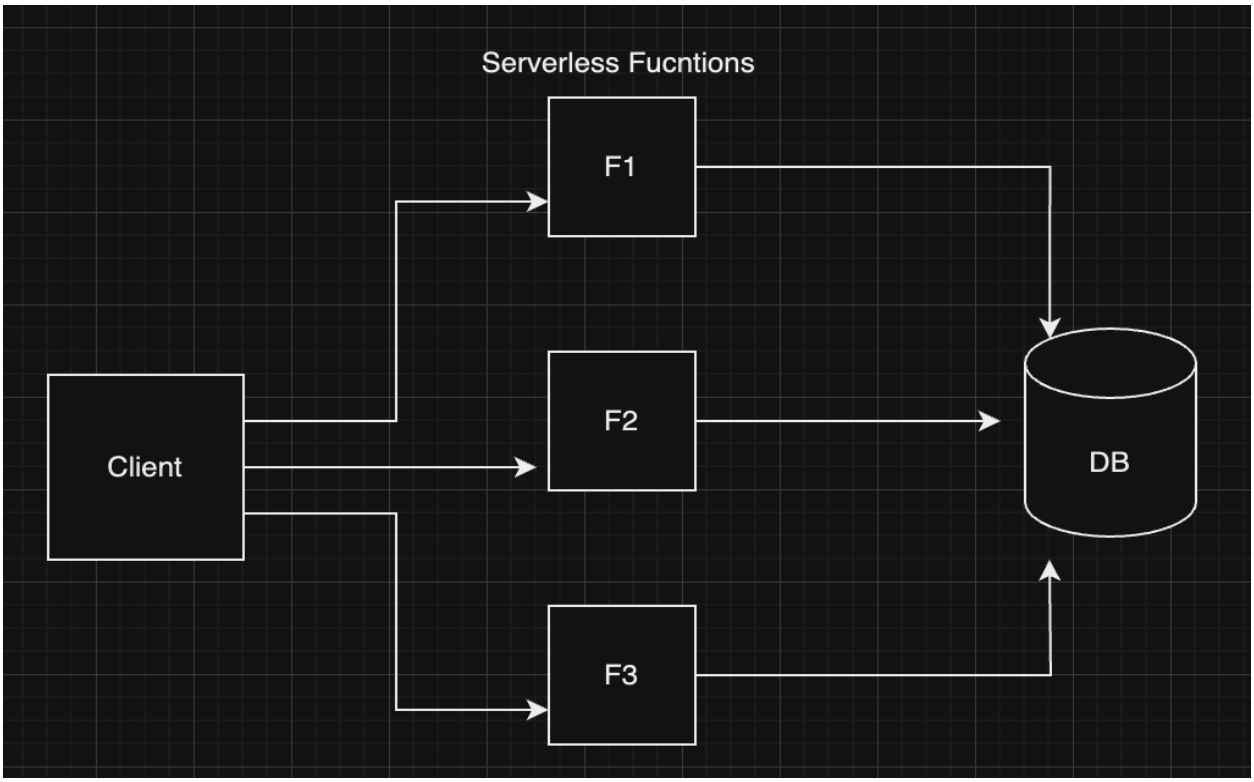

Fig 2 –Backend of a Serverless Application

Most of the attributes of a serverless function are similar when compared across different cloud providers. Key differentiating factors are the runtime memory, cost and overall adoption of the technology. Here are some of the key salient features when it comes to serverless functions -

Pay Per Use:

Serverless functions like AWS Lambda or GCF, are based on the pay-per-use model. This means users do not have to explicitly keep any of these serverless instances up anticipating traffic to their services. The functions automatically scale up based on incoming traffic, and the logic to do so can be provided by the user while doing the initial configuration of these functions.

Fully Managed Infrastructure:

Maintaining bare-metal servers on-prem or managed instances in cloud could easily ramp up costs, as well as valuable time for a developer. Serverless functions bring significant cost-cutting benefits with respect to these. All the developers need to do is perform the initial configuration of these functions with the storage, memory needed, and rest will be taken care of by the cloud provider.

Integration with Other Services:

Besides serverless functions, cloud providers offer numerous other tools like databases, object storage models like S3 buckets, file systems, memcache etc. Since serverless functions are event-driven in nature, they can easily integrate with almost every other service/tool available within a providers' ecosystem.

Invocation Methods:

Serverless functions support both synchronous and asynchronous style of invocations. Example – AWS Lambdas wait for a response in synchronous style while it is not the case in asynchronous executions.

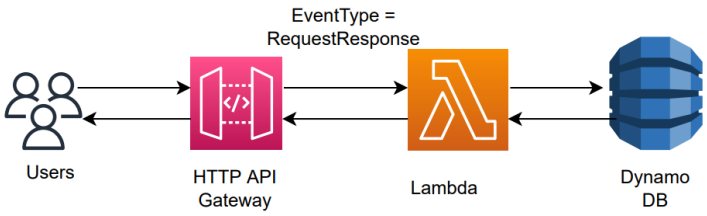

Fig 3 – Synchronous Lambda

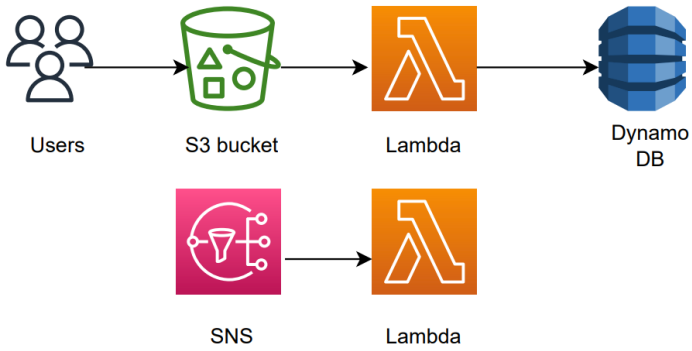

Fig 4 – Asynchronous Lambda

Cost and Time differences between AWS Lambda and GCF:

While most features are similar between both Lambda and GCF, there are some key differences primarily surrounding the cost and timeout.

Standard cost of Lambda and GCF

Lambda	1 million invocations free per month. Up to 400k GB/seconds free	Beyond free-tier $0.00001667 per GB/second
GCF	2 million invocations free per month. Up to 400k GB/seconds free	Beyond free-tier $0.0000004 per GB/second. CPU time is also calculated at $ $0.0000100 per second

Table 1 – Lambda vs GCF cost

Lambda costs are comparable to GCF. Usually, they are slightly lower when calculated overall, however GCF does allow slightly higher number of invocations which could potentially save costs.

Execution Differences

Lambda	1000 concurrent function executions, timeout 15 minutes
GCF	1000 concurrent executions, timeout 9 minutes

Table 2 – Lambda vs GCF timeout

Lambda functions clearly outshine in their execution aspect allowing far higher timeouts on a single run. This is key to running machine learning inference tasks when compared to GCF as will be evident in the experiments performed.
Besides many advantages, serverless function's do suffer from some drawbacks which are listed below -

Cold Start Problem:
Cold start as the name sounds refers to the delay in dynamic infrastructure provisioning by the cloud providers when executing any task. By default, cloud providers provision containers where these tasks are run, and cold start refers to the delay in provisioning these containers. In AWS Lambda, every function or container lasts for a duration of 15 minutes, which means it stays warm for 15 minutes. Any calls made to this function during this time, would be executed far sooner than the initial task calls. Bahar et al in [20] describe this phenomenon in the following architecture diagram.

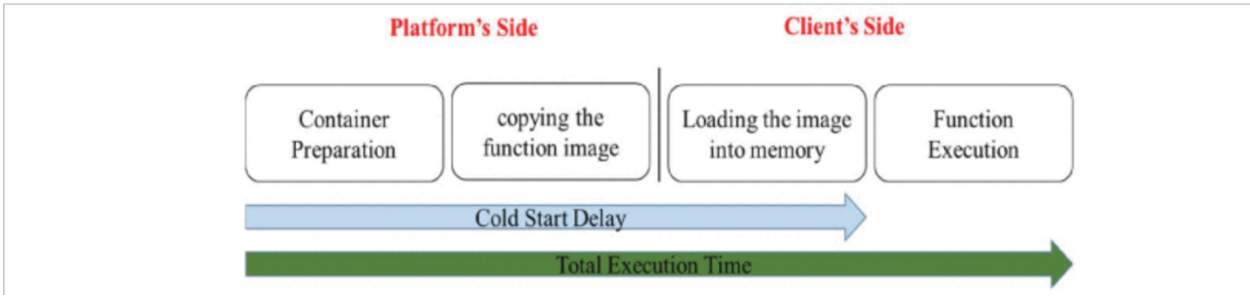

Fig 5 – Cold start delay

As can be seen, it becomes imperative that cold start delays could significantly impact the overall execution times of any task. This paper will discuss ways to mitigate this cold start problem as well and suggest an architecture using some key AWS offerings.

Besides serverless functions, another key serverless offering this paper discusses is AWS Elastic Container Service with Fargate. This paper discusses Fargate in detail and compares it with Lambda in terms of execution times. The equivalent of Fargate in a GCP is the Google Cloud Run. ECS in essence are docker containers, that can be provisioned dynamically like an AWS Lambda execution environment. It provides isolation of tasks, which is key in the architect of microservices. Since, lambdas inherently lacked capacity to run voluminous workloads, ECS becomes critical, as it provides both serverless execution and higher memory or storage. Fargate is an engine for ECS containers. It simplifies the deployment of these containers, by removing the need for developers to choose instancy type, managing scaling of these containers, or run other optimizations and maintenance tasks. Fargate ensures that developers can utilize AWS' serverless platform to the maximum benefit. The key differences between a lambda and Fargate is tabulated in the following page –

	AWS Lambda	AWS Fargate
Execution environment	Amazon Linux	User defined (more choices)
Memory	128 MB – 10GB	1GB – 30GB
CPU	AWS controlled	0.25 – 4 vCPU
Disk Space	512 MB – 10 GB	20 GB – 200 GB
Max execution time	15 minutes	No Limit
Max parallel executions	1000	500
Deployment Unit	Zipped code/container	Container

Table 3– Lambda vs Fargate

3. Related Work

Serverless computing is a brand-new framework for running machine learning models in the cloud. Most of the contemporary research focuses on improving the inference response times. This means models are trained locally on-prem or managed servers and uploaded in cloud storage. Inference tasks are then run on this uploaded model. Chahal, Ojha, Ramesh and Singhal [1] performed a performance and cost-based analysis on deploying a large Natural Language Processing model using FaaS architecture. Their evaluation of improved performance in terms of response times and cost of the FaaS platform, acts as a solid bed work for this research project. In [1], the focus has been on using AWS Lambda to run inference tasks on an Optical Character Recognition (OCR) model uploaded in cloud storage EFS. The methodologies discussed in this paper, provide good guidance as to reducing the total memory usage by an AWS Lambda function. The authors have also found a strong correlation between response time of a function and the total memory allocated to the function in run-time. Higher the memory allocated better the results, albeit at a higher cost [1]. The cost of a FaaS platform can also be deduced from most of the cloud vendor websites [2] – [5]. Overall, running inferences of 50,000,000 invocations for an AWS Lambda function with 256 MB of memory (RAM) ranges anywhere between $100-$120.

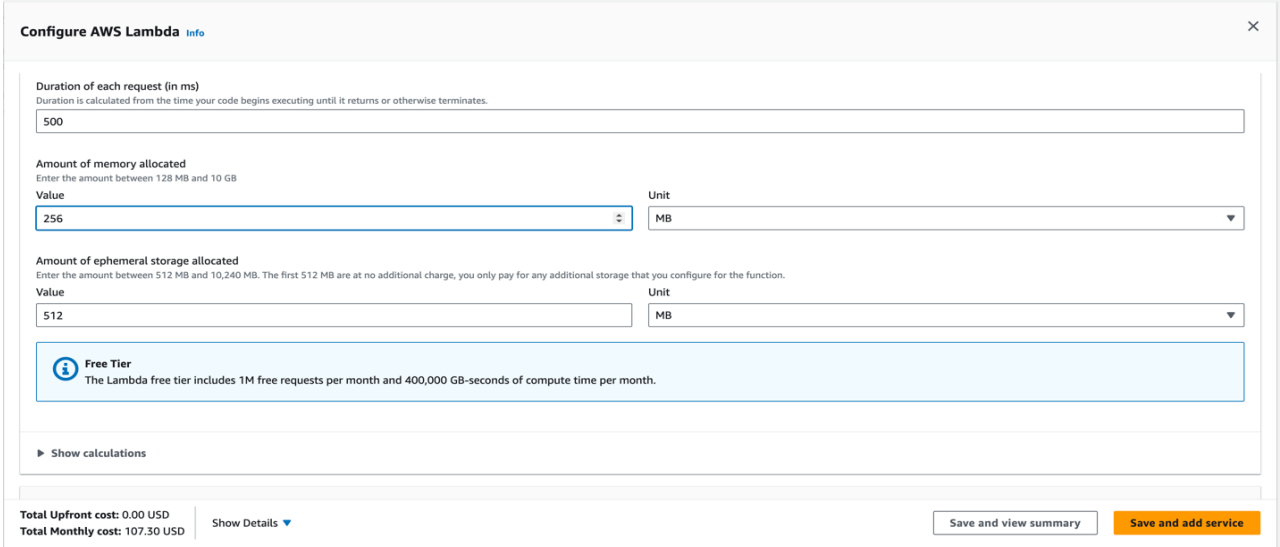

Fig 6 – Total cost of AWS Lambda (AWS Pricing Calculator)

The popularity of serverless architectures have also inspired the scientific community to port applications running on a VM to a serverless container. Poletti and Llorente [6], performed a detailed analysis between running an application, that analyzes data from the European Space Agency's Mars Express Orbiter in a serverless platform and running it in a server. Their reimagined architecture works with a Lambda that invokes a C-based executable when data is uploaded into an S3 bucket.

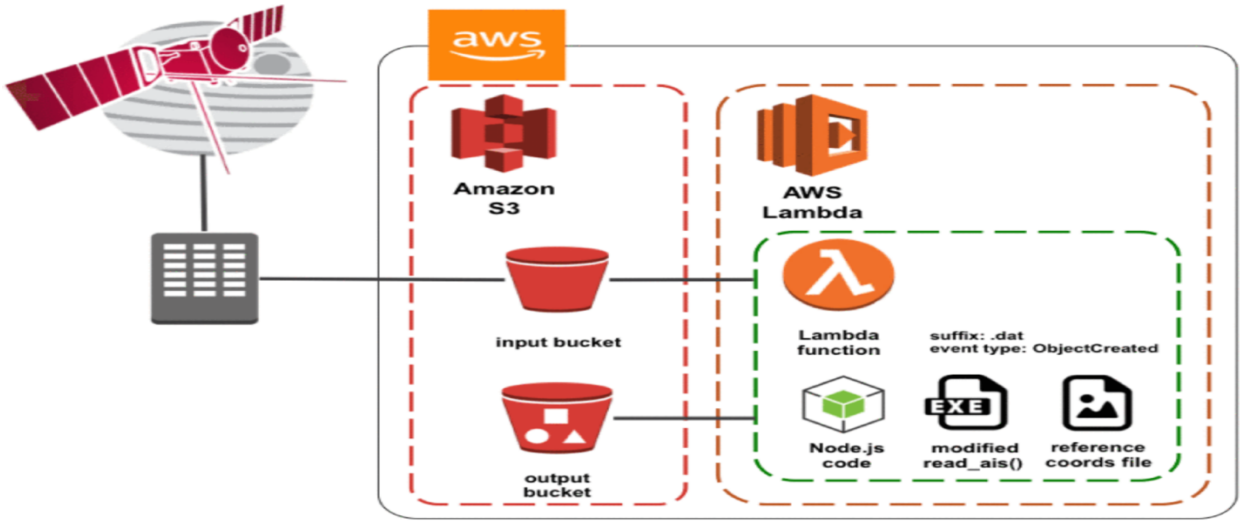

Fig 7 – Scientific workloads on AWS Lambda

This new architecture had a significant impact on the execution times of the overall inference model reducing it from 3.4k Milliseconds to 1.8k Milliseconds. For larger image files (> 17MB), the response time reduced by over 200%. This alone should serve as an inspiration for the research community to consider serverless as their go-to platform.

While example studies above dealt mostly with AWS' serverless offerings, there is also work being done on the Google Cloud Platform. Google Cloud Functions or GCF are the equivalent of Lambda functions. Perhaps the one positive differentiating factor for GCF is that they do not require attaching an API Gateway separately as they come prepackaged with an HTTP endpoint. This can potentially save us extra cost, albeit minimum. Sahar [7] created an event driven model using GCF and Google Pub/Sub architecture which is based on the Publisher-Subscriber framework. For any real time streaming of data, like a real-time weather analysis model, pub/sub architectures allow us to store events in a queue which further allows its processing in a set order. In [7], the author deploys a deep learning model packaged using Pythons' Pickle library into Google Cloud Storage. The average execution times were around 400ms, and memory utilization was around 230 MB which are below the max limits defined for the Cloud Function. The Functions did do poorly when load testing was strenuous, and this could potentially be improved using a container-based approach as we will discuss below in our experimental methodologies.

Further work of deploying deep learning models have been done by Slominski et al in [8], wherein they deployed models like TensorFlow and MXNet for image classifications which are bigger in size. They obtained better results with warm requests as compared to cold requests. The latency was around 3-4 seconds which is

expected for warm requests however, they were higher for the cold requests. The research work in [8] clearly shows a proportionality between the memory allotted to a lambda and its execution times. As an example, the authors here allotted higher 1024MB for the lambda and it performed significantly better over 512 MB for an overall model size of 100MB which would be loaded in run time. Their work does provide a baseline when it comes to tuning the parameters of a lambda environment.

Jaafar et al in [9] have studied Machine Learning Operations (MLOps) in detail based on existing research work around this area. MLOps deals with the management of machine learning pipelines and models in cloud, including their release and updates [9]. They have shown that while cold starts are a problem in serverless computing, they are still better compared to executing workloads in a managed virtual machine like EC2 instance. Research around this shows that costs of up to 2.5-3.5x can be immediately saved in maintenance of EC2 instances. Additionally, they have concluded that researchers and industry is moving machine learning pipelines to a serverless architecture primarily to obtain the benefits of the pay-per-use model. Kim et al in [10] use an event driven architecture for predicting handwriting using the MNIST dataset. They have used docker registry to store their models and dataset and used an event driven or API approach for serverless function orchestration. Their results indicate that the serverless approach can guarantee optimal response and running time, reduce the end-to-end delay of the machine learning application and shows its capability to support distributed machine learning. Sinuraya et al in [11] presents a unique case study of building a chatbot using NLP and deploying it on a serverless architecture. The chatbot performed quite well with an 89% precision score and less latency. Their model is based on webhooks and Facebook API which can be translated to AWS Lambda and API Gateway.

Coming to training machine learning models using a serverless framework. There are significant challenges to the idea of training in a serverless model, especially around the constraints of a serverless function not being intended to run intensive tasks which require lot of memory. Runtime memory is limited, maximum of 10GB which is clearly not sufficient to train most complex models. Despite these challenges, Li et al in [12] designed a novel serverless architecture called SIREN for deep reinforcement learning. They built a neural network-based job scheduler which schedules training tasks on the lambda and dynamically allots the memory and instance numbers. SIREN proves that it can reduce training time and cost by over 40% when compared to traditional training in EC2 or VM's. Klimovic et al in [13] have presented a model based on lambda called LambdaML for machine learning training. Their results also show that the response times of a serverless model is less while the costs incurred is not significantly lesser than the VM approaches (IAAS). They have presented a tradeoff between the FaaS and IaaS cloud computing architectures which is worth applying in this project as well.

Parallelizing machine learning training is also a key research agenda. Parallel hyperparameter tuning in [14] by Silva et al, shows a lot of promise in benchmarking serverless training approaches particularly running them in a distributed fashion. Data parallelism like batching a large dataset and using machine learning ideas such as mini-Batch Gradient Descent to update parameters is a key research idea used in this paper. The architecture built in [14] could be remodeled using AWS Batch jobs for orchestration and is being discussed in the later sections of this paper.

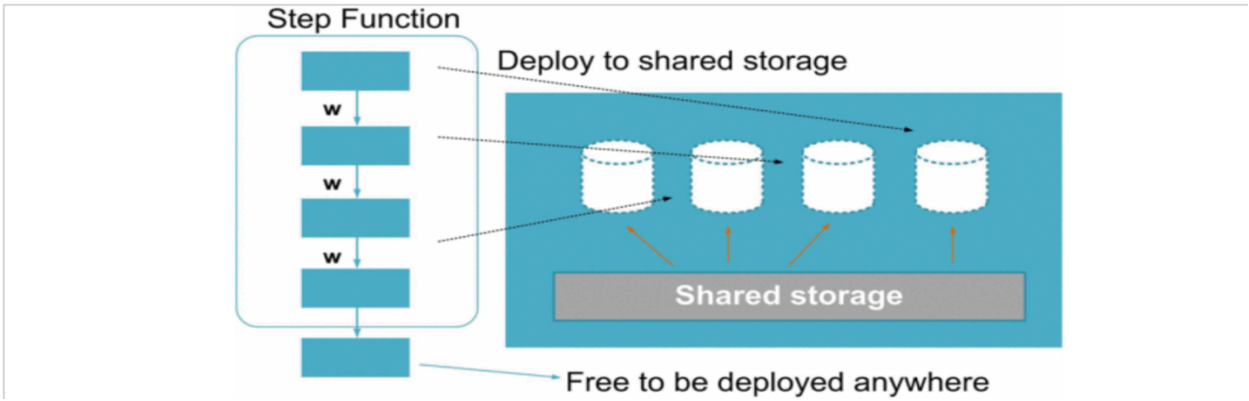

Fig 8 – Parallel machine learning using AWS Step Functions

Mishra et al in [15] also described a parallel training model using a dedicated parameter server for model parameter updates. Their concept is also based on mini-BGD approach mentioned above and looks at different data storage techniques like S3, DynamoDB, Memcache. Memcache was overall costlier however, also had the fastest training times. They also discuss the latency effects of training models and using intermediate data storage and DynamoDB seems to have the highest latency compared to S3 and Memcached. In addition to the above resources, this paper also investigates the potential impact of using a networked file system like AWS Elastic File Storage for data storage and retrieval and its effect on the response times.

4. Machine Learning

While the focus of this research is to build and compare different serverless architectures for a machine learning problem, it is also essential to talk about the different machine learning tasks undertaken to build the model.

Data Gathering

The dataset for this research was collected from open-source websites Kaggle [22] and crisisNLP [23]. It pertains to Tweets collected from Twitter which have been classified into two categories "Disaster" or "Not Disaster". The dataset consists of over 100k tweets with 4 fields namely id, location, text, and target. Each record has a unique id represented by the id field. Target labels the tweets to either of the two categories "Disasters" or "Not Disaster".

ID	Location	Text	Target
1	Sri Lanka	When volunteer orgs are more forthcoming with their transactions than some govt. offices when presented with RTI requests. #FloodSL #lka	Disaster
2	Sri Lanka	Flood Relief Food AID Collection from Muslim community in Akkaraipattu.Organized by HOPE Sri Lanka & Al Izza Hifilul Quran Girls Mathrasa	Disaster
3	Sri Lanka	Water level situation in major rivers as of 0930 hrs, 01.06.2017 - Kalu, Gin gangas still on alert - but water level falling. #FloodSL	Disaster

Fig 9 – Dataset

EDA

The training and test data consists of over 100k tweets. 43% of tweets in the training data related to disasters while the rest were non-disasters.

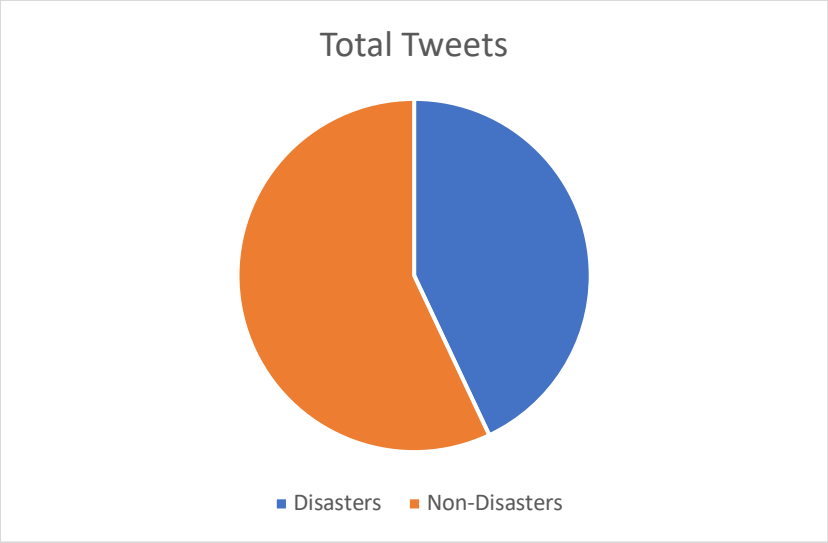

Fig 10 – Disaster vs Non-Disaster Tweets

Data Cleaning

Several data cleaning tasks were performed like removing punctuation, removing hyperlinks, numbers, and brackets from the text. Additionally, the text was transformed to lower case for consistency.

Following is a word cloud of the keywords from both disasters and non-disasters tweets.

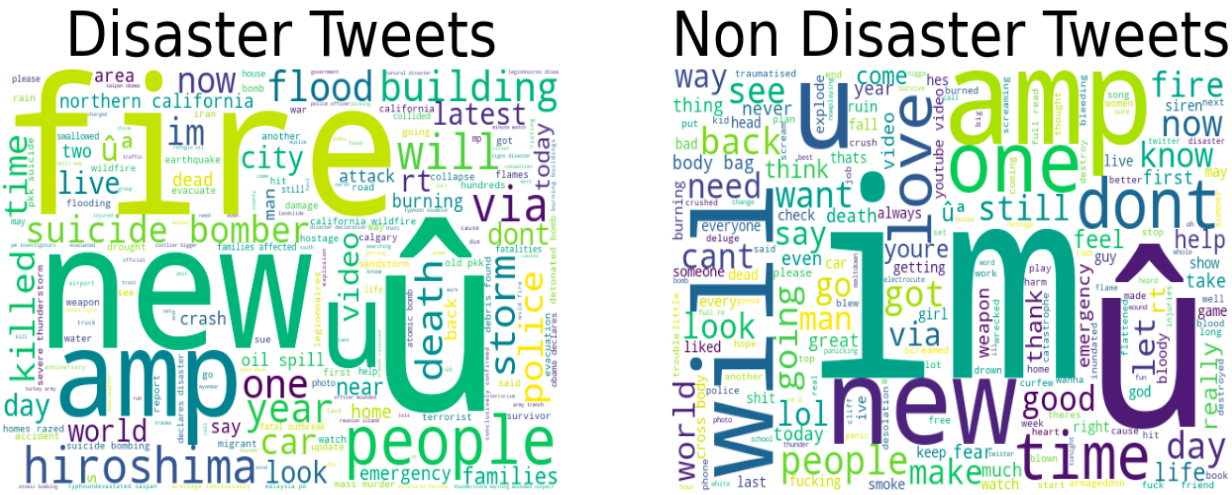

Fig 11 – Word cloud of tweets

Data Preprocessing

Tokenization – Post data cleaning, each text in the corpus was tokenized into separate strings. Each line of text was tokenized for each paragraph and each word was tokenized for each line of text.

Stopwords Removal – Later, common stop words in English like articles the, an, a etc., were removed from the tokenized text.

Stemming and Lemmatization – In this case both stemming, and lemmatization were performed on the text corpus. Stemming and lemmatization are both used as text normalization techniques. Stemming was faster in this case.

Text Vectorization – Since machine learning models take numeric inputs, the text corpus was vectorized using TF-IDF Vectorizer.

Machine Learning Model

Gradient Boosting Classifier

This research used a Gradient Boosting Classifier (GBC) model to determine whether tweets are related to catastrophes. The ability of the GBC, a potent ensemble learning method, to progressively construct decision trees, each making up for the shortcomings of the one before it, led to its selection. Using variables like location, text content, and binary indicators for disasters, our model performed well in identifying subtle patterns in the dataset. To attain the best possible model performance, hyperparameter optimization was carefully considered. The larger the depth of each tree (max_depth) and the number of estimators (n_estimators) played a crucial role in determining the complexity and size of the model. Increasing the

number of estimators resulted in a more expressive model that could capture complex patterns in the dataset, but at the cost of a larger model size. In addition, changing the maximum depth made it possible to comprehend feature interactions more deeply, which affected how interpretable the final model was. The max_depth and estimator were adjusted to ensure that larger model sizes could be obtained, as the goal of this research was not to create the most nuanced and balanced model possible. As a result, 121MB, 458MB, and 1034MB models were made. The ensuing accuracy results were as follows:

Size	Accuracy
121 MB	82.1%
458 MB	88%
1034 MB	96%

Table 4 – Comparison of size and accuracies of GBC

5. Methodologies

5.1 Inference Architectures

This section deals with experimenting with different inference architectures for the machine learning models built in the previous stage. The models were trained locally and uploaded into different cloud storage units, and their latencies in each case has been compared through load testing.

5.1.1 Single Serverless Function (Worker)

In this first experiment, the focus was to allow a single serverless function, run both as a machine learning training model, and perform inference tasks. This way a clear baseline could be set when using a single function as a worker. Research into serverless function's shows that they are only capable of running small tasks as part of the event-driven paradigm with only 15 minutes of available runtime. The analysis done concurs with this.

Case 1 – Training

In this case, a single worker (single AWS Lambda) was used for training a model discussed in 4 above. The estimators and maximum depth were kept low, since lambdas can only support up to 10GB RAM and 10GB storage. All the required data, and third-party python modules were loaded as a lambda layer. Lambda layers are a central storage for common external libraries for lambda functions, so that they can be made available in run-time. The architecture with AWS is shown in the following page.

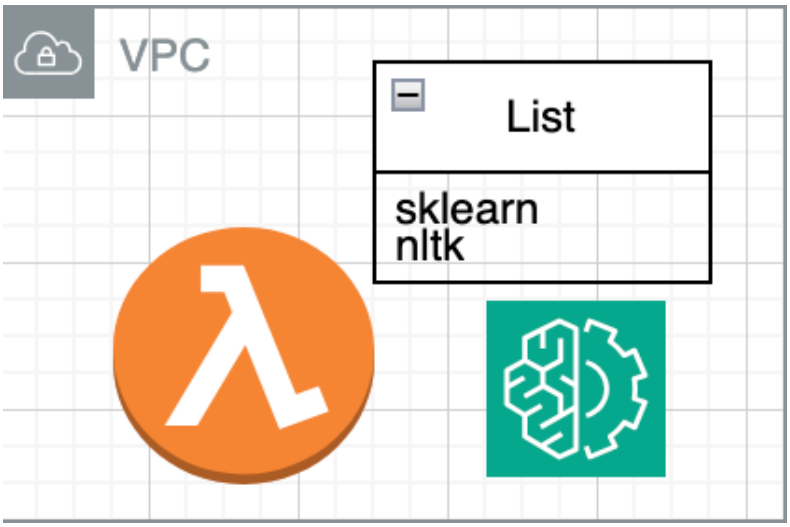

Fig 12 – Training using a single lambda.

This architecture for training a model performed rather poorly. Running with 8GB RAM, on a dataset with size 25MB, the lambda took almost 12 minutes to train. For lower RAM, it timed out in every case.

Case 2 – Inference Tasks

In this case the model and its required third-party libraries are loaded as a lambda layer in AWS. A lambda function is used to execute the inference tasks. The analysis is as shown in the below table. Higher RAM allotment results in better execution times as can be seen below.

RAM	Time (minutes)
2GB	3
4GB	1.8
8GB	0.9

Table 5 – Single Lambda for inference execution time

The table in the previous page shows that running data intensive tasks directly on a serverless function without any external storage is not ideal. For lambdas with a 2GB RAM, inference tasks took almost 3 minutes in some cases to return a result. Some of the requests were also dropped which can be seen below –

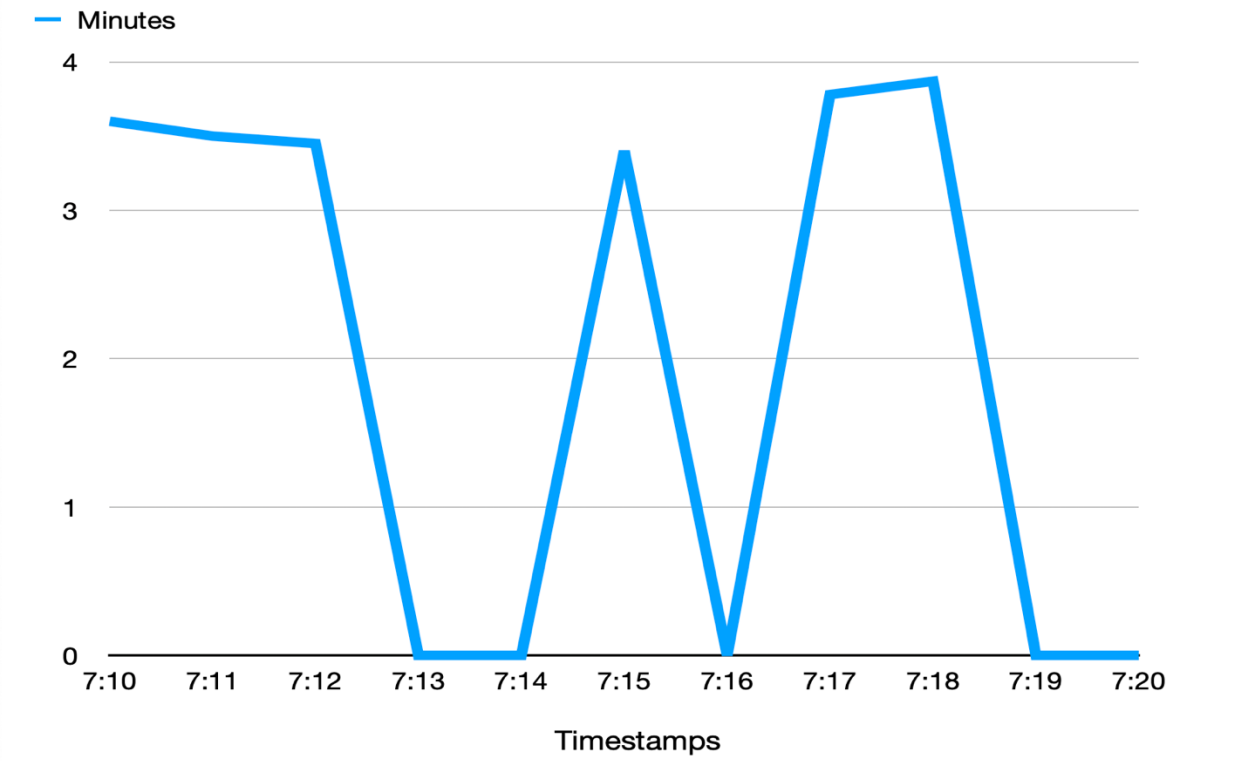

Fig 13 – Inference for a lambda

5.1.2 Serverless Functions and Object Storage

Bulk of the current research into serverless machine learning deals with this form of inference architecture. In this method, an outside bucket is used to store the machine learning model. The necessary libraries are loaded in run time as lambda layers. Most of the research [1] – [3] preferred this architecture because of its simplistic form, as well as lesser costs since an object storage model is cheap compared to other forms of storage like a database or networked file system. Its advantage over the previous architecture discussed is the freedom to store much larger models, albeit keeping the serverless function's RAM and storage in consideration.

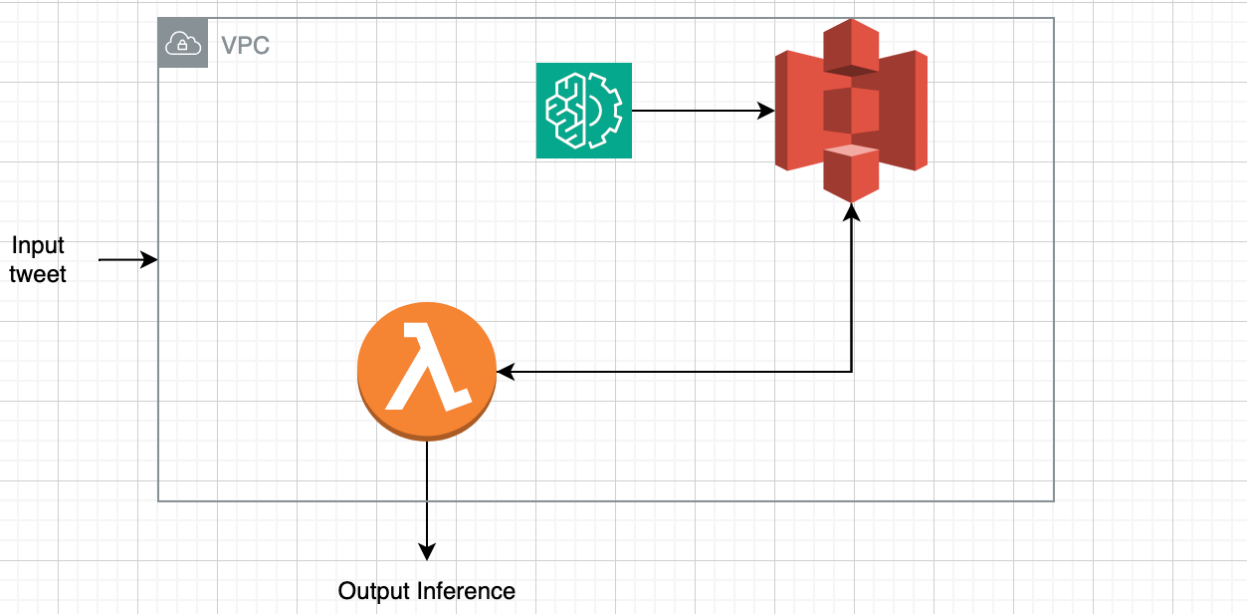

Fig 14 – S3 to Lambda

This model does however, run into cold start problems. In this research, it is shown that keeping the lambda warm (essentially keeping its state of execution running) can hold significant benefits in lowering the execution times for inference.

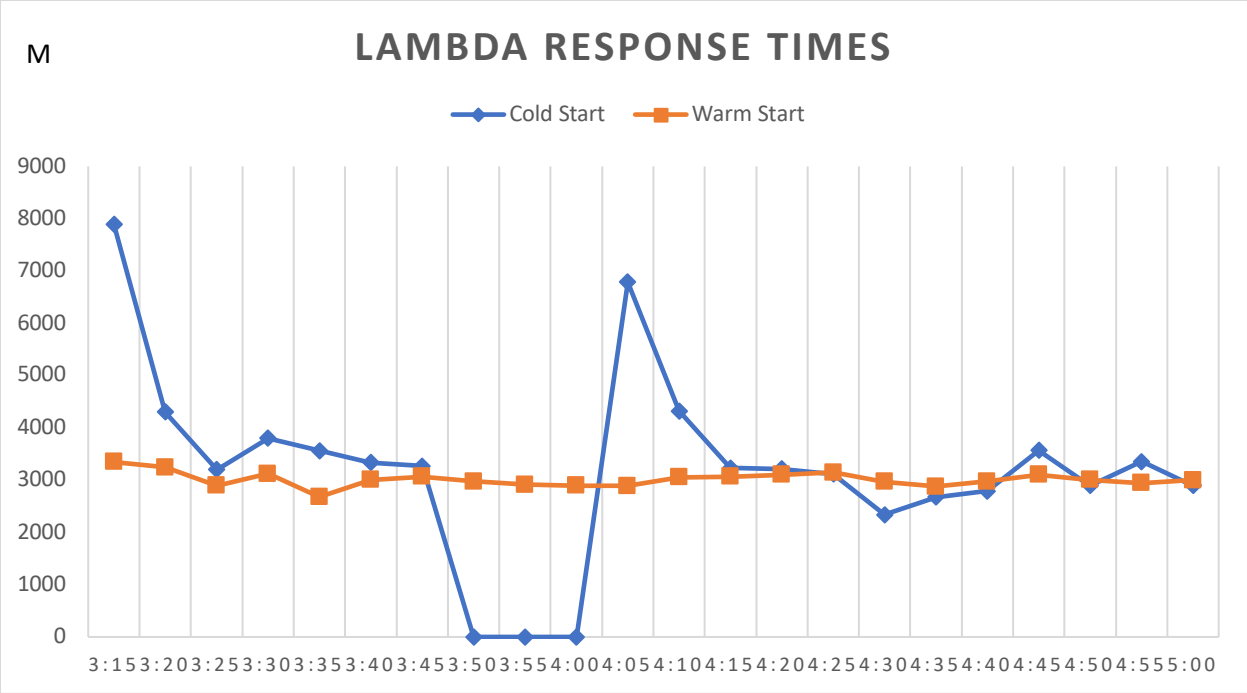

Fig 15 – Lambda and S3 response times

Different techniques can be followed for keeping a lambda warm. In this research, an AWS eventbridge rule to trigger this lambda periodically.

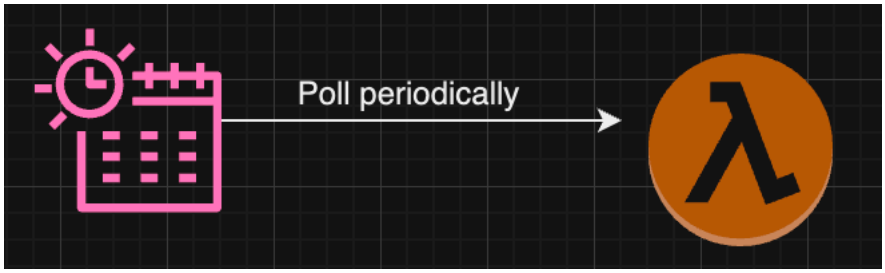

Fig 16 – Warming Lambda

5.1.3 Serverless Functions and Network File Storage (NFS)

In this section, focus was on improving the latencies seen previously in the case of object storage models. A distributed network file storage system can be easily mounted across several serverless functions at run-time. This file system can also be

shared among multiple instances or containers, and the underlying NFS protocol makes it suitable for low-latency shared access.

The architecture in this case is almost like previous use-cases, with the notable difference being the file system. In the diagram below, AWS Elastic File Storage or EFS is used in conjunction with a lambda function. The EFS stores the pre-built model, as well as other third-party libraries required by the model during run-time.

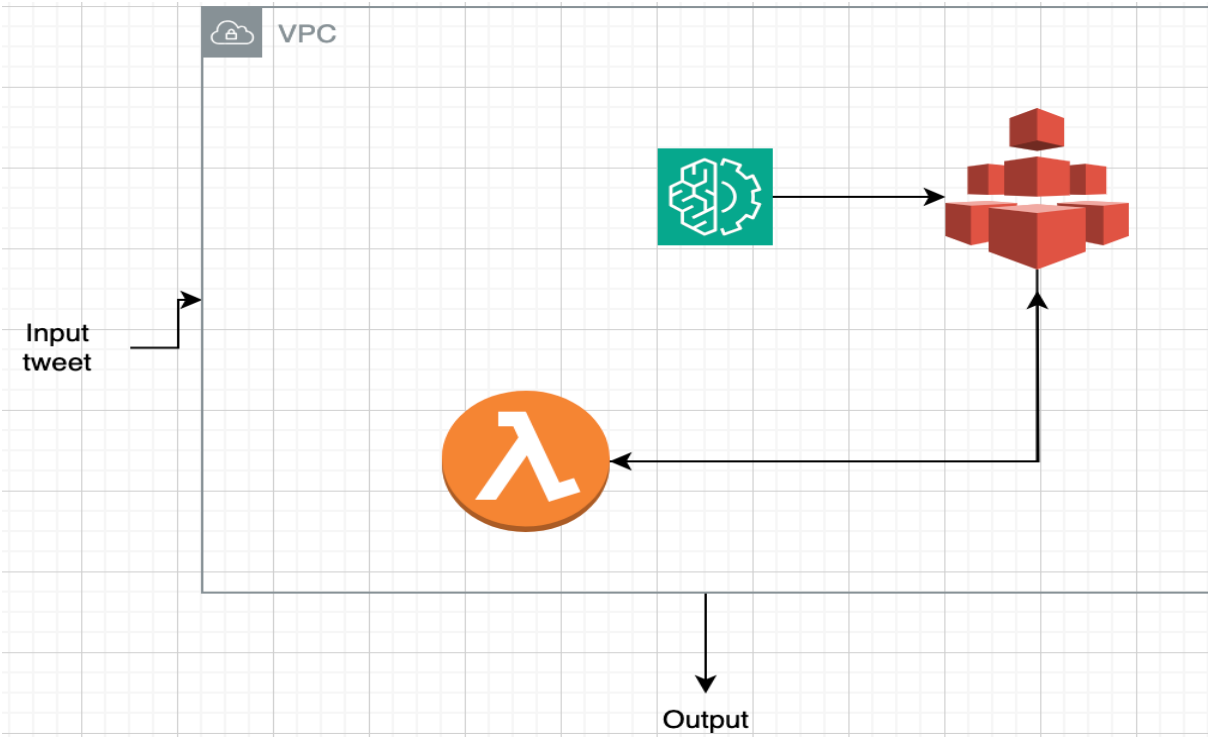

Fig 17 – Lambda and EFS architecture

The average response times significantly improve in both cold and hot start for the lambda functions. The chart in the following page highlights the same.

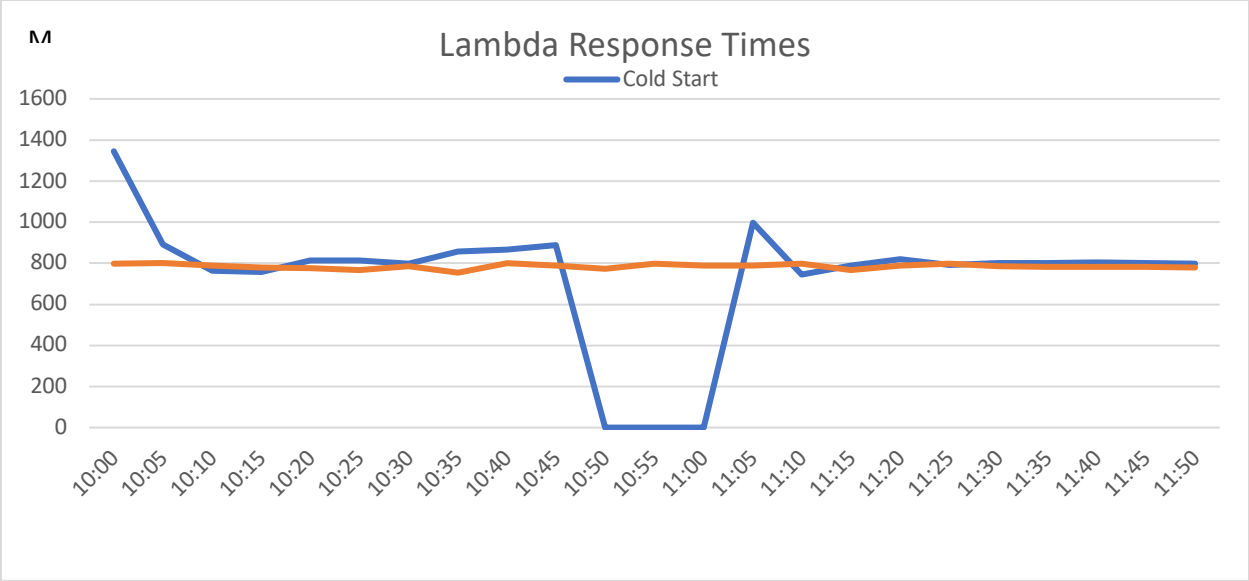

Fig 18 – Lambda and EFS response times

While this architecture performed better over the object-storage one in 5.1.3, due consideration must be put on the following points –

1. An NFS model is costlier compared to an object storage model. It is also costly for frequent access, which in this case is true. NFS like EFS in AWS will cost 2x compared to object storage S3, which virtually ends up costing nothing.

2. NFS should be preferred for lower file sizes with more frequent access requirements [21]. It should also be noted that object storage models are used for WORM (Write Once Read Many) operations, while an EFS allows frequent writes with locking which is beneficial as will be discussed in the sections below.

5.1.4 Proposed Architecture

As observed in previous sections, developers must consider trade-offs when using an object storage system compared to a distributed network file system. To tackle this uncertainty, this section investigates utilizing more recent advances in serverless container orchestration, particularly using AWS Fargate.

AWS Fargate provides a serverless compute engine to spin up ECS containers at scale on-demand. Developers are only required to provide the task definition, which includes parameters such as the type of container image (Windows, Linux, etc.), and CPU and memory requirements. The underlying orchestration maintenance of these instances is completely taken care of by the cloud provider. This type of architecture is ideal for machine learning since the primary requirement is higher computing resources like RAM and CPU. The architecture is shown in the following page. For running an inference model, a trained model is pre-built into a container image and uploaded to the Elastic Container Registry (ECR), which is a repository of AWS-hosted containers, from which Fargate spins up multiple containers based on demand and requirement. Each container then runs the inference jobs.

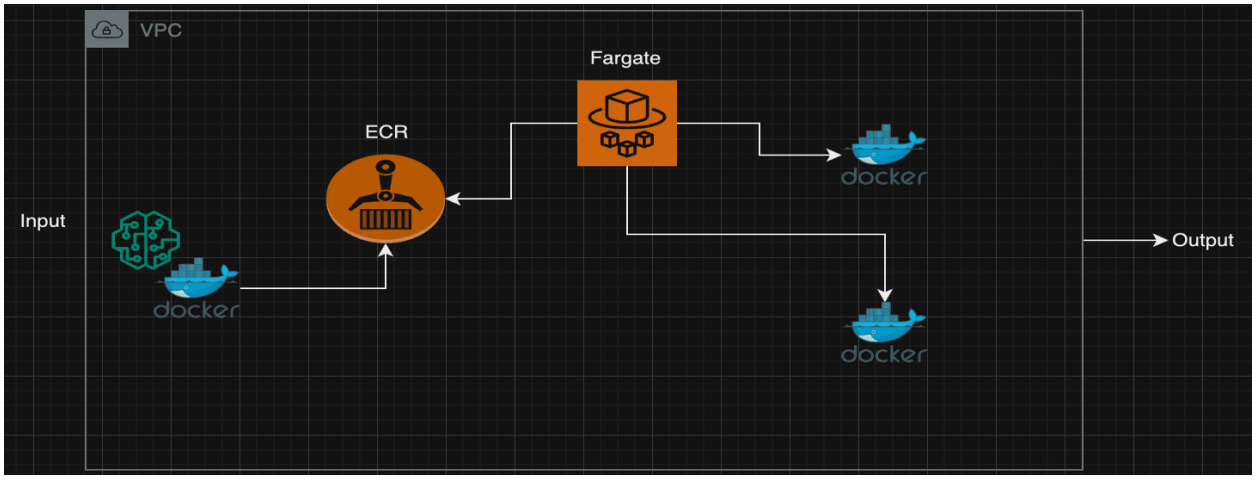

Fig 19 – ECS Architecture

Significant improvements were observed, with much better scaling of containers. The average execution times for an inference task were observed to be around 600ms for a single container instance. As the number of invocations increased, Fargate started spinning up more containers to tackle the traffic, which further reduced the response times. The following is the representation of the same analysis. With more containers available to tackle higher loads, response times are faster. This can also be configured based on the amount of CPU utilization, where if the CPU utilized in a single container is over 50%, Fargate can spin up more containers. This architecture performs significantly better than the previous ones discussed, and one of the reasons for this can be attributed to low-latency persistent storage in the case of ECS containers compared to an NFS or object storage.

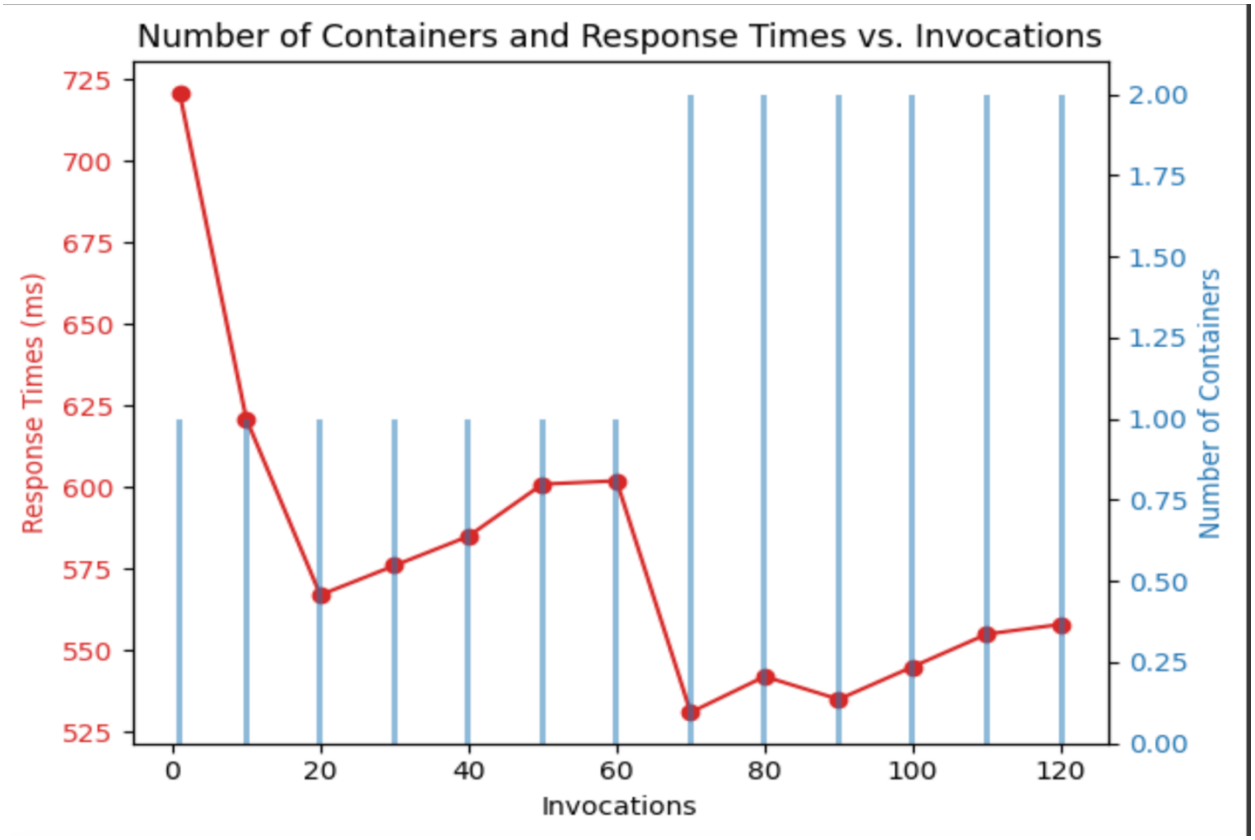

Fig 20 – ECS Response Times vs Load

5.1.5 AWS vs GCP

When it comes to GCP, it also provides similar resources for inference models like AWS as shown above. The GCP equivalent of Fargate is Cloud Run. Cloud run performed like Fargate in terms of response times of inference tasks. On average, a single container took around 700ms, and the scale-up of containers was also quite comparable to AWS Fargate. Following is the GCP architecture of the proposed model.

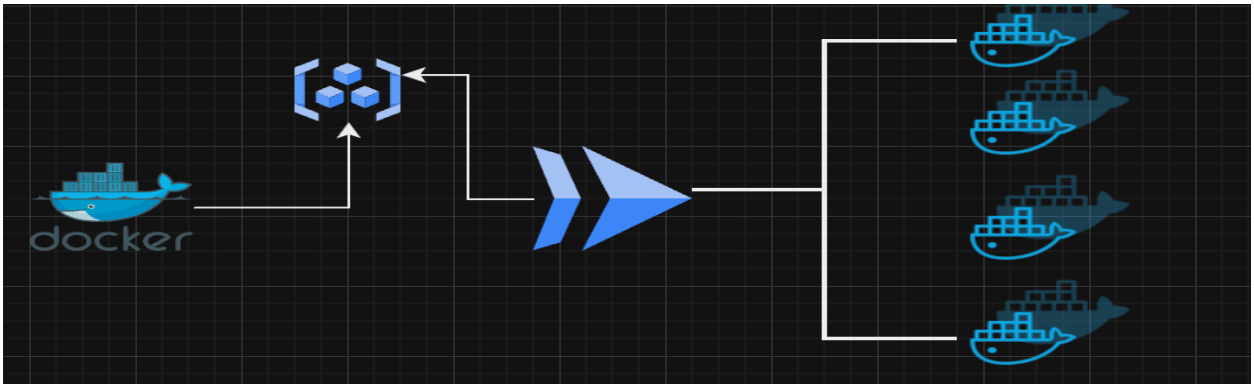

Fig 21 – Google Cloud Run

The table below highlights few key differences between GCF and Lambda functions observed during the experiments, when attached with an object storage system for access to the model and third-party libraries.

Lambda	GCF
Faster performance	Slower in experiment
Higher learning curve	Lesser learning curve
Need additional API GW attached	Packaged HTTP endpoint (simpler)
Takes 3k MS on average	Takes 5k MS on average
15 min timeout	9 min timeout

Table 6 – AWS vs GCP

5.2 Training Architecture

Data parallelism and parallel hyperparameter tuning are some of the most researched topics in machine learning. Similar ideas can be utilized in the case of training a machine learning model in serverless framework. In [6], Kudva et al showed the same using a single parameter server to store the updated hyperparameters for a neural network model. The emergence of batch framework in serverless and serverless container orchestration at-scale, particularly in AWS, is investigated here for its feasibility as a training platform.

In this architecture, AWS Batch jobs are investigated. GCP also has tools like DataPrep however, AWS Batch is cheaper and more evolved, and developers only pay for services like ECS when attached to AWS Batch. There is no charge for using Batch itself as it facilitates executing tasks in batches on containers. This simplifies the machine learning training process, because now data can be batched prior to feeding into a container for training. Batching data reduces the requirement to pay for greater computing resources like RAM or vCPU for a single instance. Combining these batch results for purposes like hyperparameter tuning or model update itself could significantly reduce the overall training duration using a serverless framework. The alternative is to train in-silo on a single instance, while batching allows to map and combine tasks for a more nuanced model.

In this experiment, an attempt is made to understand the feasibility of parallelizing data for hyperparameter tuning using different AWS services. The steps are listed on the following page –

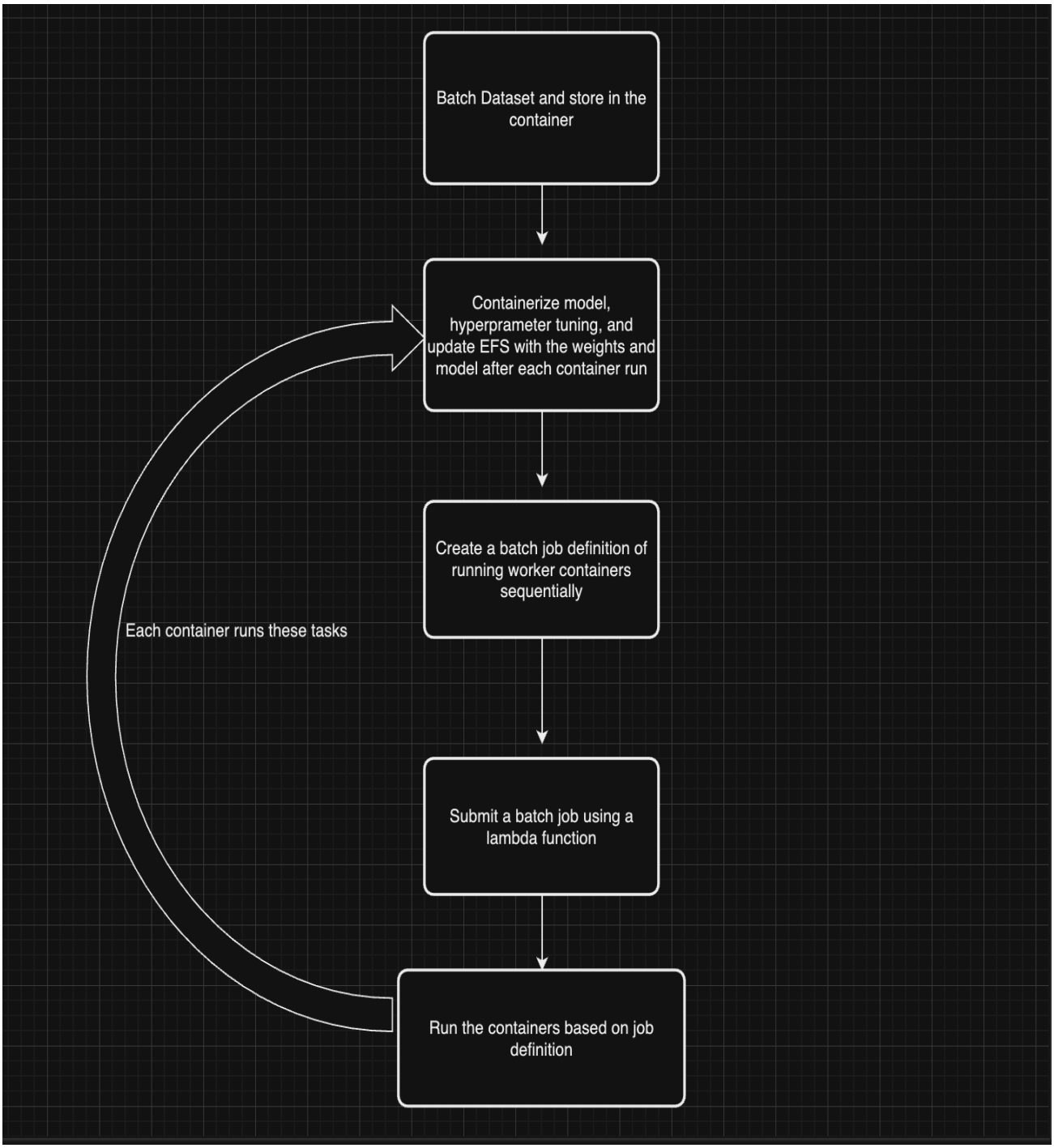

Fig 22 – Training steps in serverless framework

The architecture is shown in the following page -

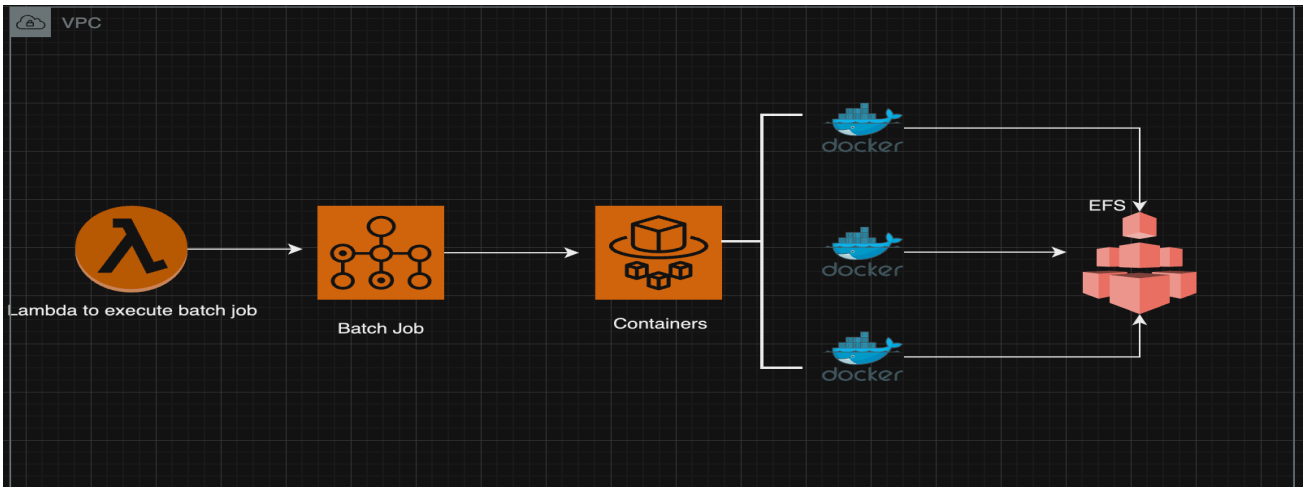

Fig 23 – Training architecture for serverless framework

Analysis of the runtimes of the model is presented in the below table. As evident, costs are the bare minimum while training a model, given the computing power allotted. It is certain that for much larger use cases, more computing needs to be made available for lower train times.

Experiment	Batch Size	#of Iterations	H/W	Time	Cost (Only containers)
1	½ of total tweets	50	1vCPU, 2GB memory	~ 75 minutes	~ $4/hr
2	¼ of total tweets	50	1vCPU, 4GB memory	~ 35 minutes	~$4/hr
3	1/5 of total tweets	50	2vCPU, 4GB memory	~ 18 minutes	~ $6/hr

Table 7 – Analysis of serverless training

6. Results

Based on the methodologies discussed above, the key results for this research are listed below in two categories – Inference and Training

Inference:

For inference-based architectures, the serverless framework can significantly reduce the cost of running simple event-driven responses for a machine learning model. Lambda functions or GCF could be used in conjunction with external storage systems, such as Network File Storage EFS, to provide a low latency-based scalable platform. However, in this research, based on response times and cost to operate, it can be concluded that running serverless containers like ECS on Fargate can be better. Additionally, it allows far greater computing for at-par or lower costs than a serverless function itself.

Training:

Training machine learning models using a serverless architecture holds numerous challenges. The key challenge is to combine training outputs for hyperparameter tuning or generating ensemble models. In this research, AWS Batch was investigated to create a data parallel platform to run training tasks. From the results obtained, allotting a greater number of containers with around 4GB of memory holds strong potential to reduce overall training duration. This is key when considering larger models like image recognition or large language models.

7. Future Scope

The research offers more possibilities for investigation in the future. First, by running resource-intensive jobs using serverless frameworks, such as image recognition models, there is a chance to improve their availability. This change reduces the expenses of using traditional servers while relieving developers of infrastructure management, which could result in more effective workflows.

Architectures for parallel hyperparameter adjustment in serverless frameworks are a further line of inquiry. Although it looks promising for improving model performance, merging model weights without running into the risk of accidental overrides is challenging. Furthermore, there is room to investigate if models or data may be stored in-memory caches offered by certain cloud providers, which offers a possible way to improve system performance in general. The efficiency and capacities of serverless architectures in machine learning applications could be improved in these future directions.

8. Conclusion

In conclusion, integrating machine learning with serverless computing significantly improves the effectiveness of training, scalability, and deployment of models. A serverless framework's capability to scale dynamically in response to variations in demand while supplying a significant amount of processing capacity is invaluable for carrying out inference operations and training large-scale machine learning models.

While there are obvious shortcomings, such as the computational constraints of serverless frameworks, research into distributed serverless architectures is prompted by these challenges. This provides opportunities to address and mitigate these constraints, enabling the large-scale application of the fundamental concepts presented in this paper, especially when training more complex machine learning models. A new era of efficient and scalable applications in the field is about to begin, thanks to the potential that the combination of machine learning and serverless computing is poised to unlock.

References

[1] D. Chahal, R. Ojha, M. Ramesh and R. Singhal, "Migrating Large Deep Learning Models to Serverless Architecture," 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Coimbra, Portugal, 2020, pp. 111-116, doi: 10.1109/ISSREW51248.2020.00047.

[2] AWS Lambda Functions, 2023, [online] Available: https://www.amazon.com/lambda/

[3] IBM Cloud Functions, 2023, [online] Available: https://cloud.ibm.com/fiunctions

[4] Azure Functions, 2023, [online] Available: https://azure.microsoft.com/en-us/solutions/serverless/

[5] Google Cloud Functions, 2023, [online] Available: https://cloud.google.com/serverless

[6] J. L. Vázquez-Poletti and I. M. Llorente, "Serverless Computing: From Planet Mars to the Cloud," in Computing in Science & Engineering, vol. 20, no. 6, pp. 73-79, 1 Nov.-Dec. 2018, doi: 10.1109/MCSE.2018.2875315.

[7] S. Jambi, "Serverless Machine Learning Platform: A Case for Real-Time Crisis Detection over Social Media," 2022 Second International Conference on

Computer Science, Engineering and Applications (ICCSEA), Gunupur, India, 2022, pp. 1-6, doi: 10.1109/ICCSEA54677.2022.9936459.

[8] V. Ishakian, V. Muthusamy and A. Slominski, "Serving Deep Learning Models in a Serverless Platform," 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, USA, 2018, pp. 257-262, doi: 10.1109/IC2E.2018.00052.

[9] A. Barrak, F. Petrillo and F. Jaafar, "Serverless on Machine Learning: A Systematic Mapping Study," in IEEE Access, vol. 10, pp. 99337-99352, 2022, doi: 10.1109/ACCESS.2022.3206366.

[10] T. P. Bac, M. N. Tran and Y. Kim, "Serverless Computing Approach for Deploying Machine Learning Applications in Edge Layer," 2022 International Conference on Information Networking (ICOIN), Jeju-si, Korea, Republic of, 2022, pp. 396-401, doi: 10.1109/ICOIN53446.2022.9687209.

[11] E. Handoyo, M. Arfan, Y. A. A. Soetrisno, M. Somantri, A. Sofwan and E. W. Sinuraya, "Ticketing Chatbot Service using Serverless NLP Technology," 2018 5th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), Semarang, Indonesia, 2018, pp. 325-330, doi: 10.1109/ICITACEE.2018.8576921.

[12] H. Wang, D. Niu and B. Li, "Distributed Machine Learning with a Serverless Architecture," IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, Paris, France, 2019, pp. 1288-1296, doi: 10.1109/INFOCOM.2019.8737391.

[13] Jiawei Jiang, Shaoduo Gan, Yue Liu, Fanlin Wang, Gustavo Alonso, Ana Klimovic, Ankit Singla, Wentao Wu, and Ce Zhang. 2021. Towards Demystifying Serverless Machine Learning Training. In Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21). Association for Computing Machinery, New York, NY, USA, 857–871. https://doi.org/10.1145/3448016.3459240

[14] L. Feng, P. Kudva, D. Da Silva and J. Hu, "Exploring Serverless Computing for Neural Network Training," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2018, pp. 334-341, doi: 10.1109/CLOUD.2018.00049.

[15] D. Chahal, M. Mishra, S. C. Palepu, R. K. Singh and R. Singhal, "Pay-as-you-Train: Efficient ways of Serverless Training," 2022 IEEE International Conference on Cloud Engineering (IC2E), CA, USA, 2022, pp. 116-125, doi: 10.1109/IC2E55432.2022.00020.

[16] T. P. Bac, M. N. Tran and Y. Kim, "Serverless Computing Approach for Deploying Machine Learning Applications in Edge Layer," 2022 International Conference on Information Networking (ICOIN), Jeju-si, Korea, Republic of, 2022, pp. 396-401, doi: 10.1109/ICOIN53446.2022.9687209.

[17] Eoin Shanaghy; Peter Elger, AI as a Service: Serverless machine learning with AWS, Manning, 2020.

[16] D. Chahal, M. Mishra, S. C. Palepu, R. K. Singh and R. Singhal, "Pay-as-you-Train: Efficient ways of Serverless Training," 2022 IEEE International Conference on Cloud Engineering (IC2E), CA, USA, 2022, pp. 116-125, doi: 10.1109/IC2E55432.2022.00020.

[17] A. Kaplunovich and Y. Yesha, "Automatic Tuning of Hyperparameters for Neural Networks in Serverless Cloud," 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020, pp. 2751-2756, doi: 10.1109/BigData50022.2020.9378280.

[18] M. Kiran, P. Murphy, I. Monga, J. Dugan and S. S. Baveja, "Lambda architecture for cost-effective batch and speed big data processing," 2015 IEEE International Conference on Big Data (Big Data), Santa Clara, CA, USA, 2015, pp. 2785-2792, doi: 10.1109/BigData.2015.7364082.

[19] Qian, L., Luo, Z., Du, Y., Guo, L. (2009). Cloud Computing: An Overview. In: Jaatun, M.G., Zhao, G., Rong, C. (eds) Cloud Computing. CloudCom 2009. Lecture Notes in Computer Science, vol 5931. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-10665-1_63

[20] P. Vahidinia, B. Farahani and F. S. Aliee, "Cold Start in Serverless Computing: Current Trends and Mitigation Strategies," 2020 International Conference on Omni-layer Intelligent Systems (COINS), Barcelona, Spain, 2020, pp. 1-7, doi: 10.1109/COINS49042.2020.9191377.

[21] Harisree, L. B. Rao, S. Nikhil and R. Srinivasan, "Relevancy of Replacing NFS with Storage Buckets in EDA Industry," 2022 IEEE Women in Technology

Conference (WINTECHCON), Bangalore, India, 2022, pp. 1-4, doi: 10.1109/WINTECHCON55229.2022.9832296.

[22] Kaggle for data analysis, 2023, [online] Available https://www.kaggle.com/

[23] CrisisNLP for data, 2023, [online] Available https://crisisnlp.qcri.org/