RingTMP: Locally Distributed Machine Learning on Low Power Devices

Prifysgol Abertawe
Swansea University

Christopher Hopkins

Department of Computer Science
Swansea University

This dissertation is submitted for the degree of
Bachelor

November 2, 2020

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 40,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 100 figures.

Christopher Hopkins
October 2020

Abstract

More data than ever is being produced by low power devices such as smart phones and Internet of Things (IoT) devices at the network edge. The data being produced is so enormous it would be infeasible to send it to a centralised location. Instead models can be trained from data distributed across multiple edge nodes, with machine learning algorithms being performed locally. In this paper we explore training multiple low power devices using a new distributed machine learning paradigm *RingTMP*. This paradigm focuses on low power usage and power efficiency while having the capacity for larger models than comparative systems.

Contents

1 Introduction

1.1 Motivation and Context

Machine Learning algorithms have become ubiquitous in modern life. Powering social media feeds, email spam filters, advertising personalisation and even identifying breast cancer more accurately than doctors. [1] To train these Machine Learning algorithms large datasets are needed. The greater the available data, the more nuanced and accurate the produced model will be.

As the scale of problems we are trying to solve dramatically increases, the scale of datasets are too. Services like the Internet Archive as of 2020 contain over 70PB in it's database. Labeled datasets such as AViD have video data in the order of terabytes. [2] Since 2008 Google has been processing more 20PB of data a day using their MapReduce algorithm. [3]. More recently the cutting edge GTP-3 Natural Language Processing model contains 175 Billion parameters. [4] And efforts are being made to create models with trillions of parameters. [5] Both the size of the datasets and the size of the models are being pushed to their extremes.

Deriving meaning from these vast quantities of data and obtaining nuanced insights from them is a difficult task. Not only because deeper insights into data require larger Machine Learning models. But because more data is needed to populate the parameters of these models accurately. Both of these factors contribute to the need to distribute the computation of the model and sometimes the model itself across multiple nodes otherwise known as Distributed Machine Learning. Distributed machine Learning is often a pre-requisite for training models, now that datasets and models are becoming so large and no single machine is powerful enough to process these datasets in a viable amount of time. [6]

There are many challenges a distributed machine learning system can introduce, such as scalability. As more machines are added to a system the communication overhead increases. Communication bandwidth is a limited resource which can produce bottlenecks when the network becomes saturated. The goal of a good distributed machine learning System is to have the number of nodes be as directly proportional to rate of model training as possible.

Resiliency is another factor to consider. With distinct machines working in conjunction with each other and training that is prohibitively expensive, its important that the training of a model doesn't fail. When a machine in the system fails to work a distributed machine learning system must have contingency plans. These plans can ensure the minimum amount of model data is lost, and that the machine learning can continue swiftly after a major error.

The popular current solution is to use a parameter server model. In brief the paradigm is made of two different types of components. The parameter server and the workers. The parameter server holds the global parameters of the model. Workers are given the model parameters by the parameter server. The workers then perform an iteration of whichever machine learning algorithm they are performing, modifying the parameters. Then the modified parameters are sent to the parameter server where they are aggregated, the global model parameters are updated and the cycle continues until the model has converged on an answer.

However this model is not perfect and has 3 considerable drawbacks:

- Workers either spend too much time being idle or perform redundant computation, even when the system is under full load. [3]
- Every worker must communicate with a single parameter server, this limits scalability as eventually the network bandwidth becomes saturated severely impacting performance. [6]
- Many parameter server models require the whole model to be replicated within each node. [7] This means that very large models simply cannot run on machines with low system memory.

In previous years distributed machine learning algorithms have been focused on high powered servers with large bandwidth connections. More recently there has been more focus on on-device machine learning. Work has also been done in distributed machine learning at the edge of internet networks. [8, 9] Mobile and Internet of Thing (IoT) devices utilising their power to train networks locally, rather than on a centralised server. This has been driven by the amount of data being generated on devices, its no longer viable for it all to be sent to a central server. [10] This introduces its own problems, while the total computation is potentially limitless, the bandwidth between the devices is small. Not to mention the devices have limited storage, memory and computational power. The distributed algorithms in this field have been adapted from the ones used in centralised distributed machine learning servers. It remains to be seen how well suited these are for low power distributed computing.

1.2 Aims

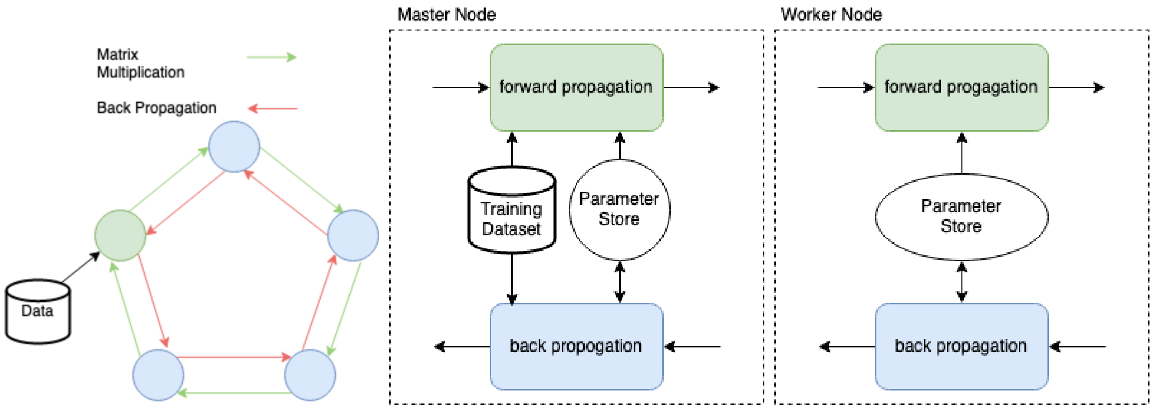


Figure 1: Left: An example network of RingTMP. The green node being the master node and the blue nodes being the worker nodes. Centre: The architecture of the master node. Right: The architecture of the worker node.

My solution to address these issues raised above and a number of others is to introduce a new model for distributed machine learning: *RingTMP*. RingTMP (Ring Topological Model Parallel) is a Ring Topological Model Parallel distributed machine learning framework focusing on optimising Distributed Stochastic Gradient Descent in low power hardware. This is a novel design drawing in inspiration much research but particularly from the STRADS and DistBelief machine learning frameworks. [11, 12]

As I have already alluded to this will take the form of a framework that will operate on multiple machines which henceforth I will describe as nodes. Each nodes contains a series of adjacent of Deep Neural Network (DNN) layers. The nodes are arranged in a ring. In the clockwise direction matrix multiplication is performed and in the anticlockwise direction gradient descent through backpropagation is performed. There are two kinds of nodes master nodes and worker nodes. Master nodes have direct access to the training data being input into the Neural Network, whilst also performing operations on the data flowing through the neural network. Worker nodes simply perform matrix multiplication and backpropagation as it is directed via adjacent nodes. There is only one master node per network and as many worker nodes as desired.

These are the benefits my system will have over existing parameter server architectures:

- Workers will have less idle time, because the work will be distributed more proportionally between nodes. And computing resources will be used more efficiently.

- There will less communication across nodes as the parameter server so not need to send their local weights to the global parameter server each iteration.
- There will also be a potential for larger communication bandwidth, as with RingTMP ring topology only adjacent nodes need to communicate. While with a parameter Server model every worker needs to communicate with the Parameter server at the end of each iteration, which can lead to bottlenecking.
- The nodes require no scheduling as in some model parallel systems. Instead scheduling is managed in a decentralised manner via communication with adjacent nodes. This also makes the system more resilient, in the case of the scheduler crashing.

My aims more specifically for this project are to:

- Create a prototype RingTMP framework and run it multiple machines simultaneously.
- Create a parameter server model framework using the same tools that will run over multiple machines simultaneously.
- Show that RingTMP reduces the time workers are idle in comparison to the parameter server model.
- Demonstrate that RingTMP takes less overall power to run in comparison to the parameter server.
- Display that RingTMP is at least as scalable than a generic parameter server.
- Show that this solution can be as resilient as a parameter server model distributed neural network.
- Demonstrate that RingTMP can hold larger models on comparison to a standard parameter server, where each worker hold the whole model.

I will do this by running both systems on low power devices, then running controlled experiments on them to measure their performance.

1.3 Overview

This document is split up into the following sections:

- **Introduction** Current section. Introduce the project and its aims.
- **Background Research** Presents related research material and similar applications and areas.
- **Project Plan and Time Management** Gives an outline of my project timeline, methodology and assessment of risk.
- **Conclusion** Summary of previous sections.

2 Background Research

2.1 Brief Introduction to Machine Learning, Neural Networks and Stochastic Gradient Descent

To first understand Distributed Machine Learning you must first understand the fundamentals of Machine Learning and Neural Networks.

There are many machine learning methods some requiring training data which we call supervised and some being able to find patterns in data without being given solutions called unsupervised. [13] An example of a supervised system may be predicting house prices, using multiple factors about each house (market data, geographic area, square footage etc.) to come to a conclusion about what the house could sell for. This would be trained using data of previously sold houses to predict current ones. An example of an unsupervised system could be identification of new plant species. This could be done by taking as many features of a plant as possible, then apply a clustering algorithm to see if there are two distinct clusters in the data. If there were then that would suggest two different plant species. Neural Networks tend to focus on supervised learning and use a form gradient descent called Stochastic Gradient Descent.

Many Machine Learning algorithms use a cost function to measure how well or badly they are solving a problem, these algorithms use parameters which are internal variables of a machine learning model and define how they solve the problem. If you map $costFunction(x)$, where x is the model parameter, for every x value. Then a graph will be produced $y = costFunction(x)$, the lowest point on the graph will be the global minimum. There may be other troughs higher than the global minimum these are called local minimums. A global minimum represents the lowest value of the cost function which indicates the parameter values produce the best solution for your problem. Initial model parameters are often randomised, meaning they may start at a high point on the cost function graph, the goal is to get to the lowest point possible. To do this you must *descend* down the *gradient* to a local minima, the algorithm that does this is called gradient descent for that very reason. This often happens in little steps after the observation of each piece of data. However it is computationally expensive to step down the gradient after each example. It is more efficient to calculate the average step of a randomised selection of data. This is know as Stochastic

Gradient Descent.

Neural Networks are structures that can perform multi-variable gradient descent when provided with training data. Neural Networks are comprised of layers of interconnected neurons in a lattice like structure. Each neuron holds parameter information the adjusting of which through stochastic gradient descent leads to the solving of a problem through reaching the local minimum of the cost function.

2.2 The Communication Issue

Distributed Neural Networks must communicate with each other in some way in order to work together. This needs to be formalised to be able to measure the efficiency of our machine learning system. Parts of this reasoning already appears in these papers too. [8, 14]

If we consider how a neural network operates if we were to run it on a single node, we could characterise its computation as such:

$$TIME = I_A(\epsilon) \times T_A \tag{1}$$

Where $I_A(\epsilon)$ is the number of iterations of the algorithm A it takes to reach accuracy ϵ and T_A is the time of each iteration of the algorithm. Here maximising the convergence per iteration or decreasing the time an iteration takes will decrease the runtime of the algorithm. In a distributed setting this equation changes to this:

$$TIME = I_A(\epsilon) \times (c + T_A) \tag{2}$$

In this equation we have the added variable c, this represents time taken for communication per iteration. In a distributed setting this will always remain above a non trivial amount of data. Unfortunately The majority of machine learning algorithms use a stochastic method which means a very large number of iterations ($I_A(\epsilon)$) that take a short amount of time (small T_A). You can see that no matter how small c is there will be a significant impact of the time taken. In fact with a naive approach of communication each iteration almost certainly $c > T_A$.

However this view doesn't take into account the possibility that communication could happen at the same time as an iteration. For example imagine a pipeline of nodes where each nodes performs an iteration but can communicate its previous iteration to the next node at the same time. Then the time taken could be described like so:

$$TIME = I_A(\epsilon) \times max(c, T_A) \tag{3}$$

Here you can see that if you can find a way of making the communication time equal to the time per iteration. Then c would have a negligible effect on the equation.

2.3 Limited History of Distributed Machine Learning

One of the first pieces of research into Distributed Machine learning was 'Distributed Inference for Latent Dirichlet Allocation' in 2008 [15] One of the first instances of DistributedMachine Learning was used to categorise New York Times articles using Latent Dirichlet Allocation (LDA), which identifies the affiliations words have to certain topics. While the paper focused on parallelising the algorithm and running them over multiple artificially isolated cores the results showed that distributed machine learning could have scalability and didn't impact the rate of convergence of the model significantly. This was followed by a paper by Jia et al. [16] which produced much faster results than its predecessors by using memcache layer in every machine, every machine would message every other machine with updates of its local parameters to create an approximate global state, it was mentioned in passing that arranging the nodes in a star topology and caching the values that passed through it could make the system more scalable. After this followed a cambrian explosion of work in this area [3, 12, 17, 18] culminating in 2014 when the parameter server as it is known today [6] was produced. This parameter server is highly sophisticated and flexible accommodating the difference in hardware components while spending more on computation and less time waiting.

2.4 Model and Data Parallelism

When creating distributed machine learning models there two different methods for distributing training, Model Parallelism and data parallelism. These two methods are not mutually exclusive and can be used in conjunction with one another, such as in DistBelief. [12]. Model Parallelism is when model parameters are split between the nodes. As Data Parallelism is when the data is split between the nodes. [19] Often with model parallelism the whole set of

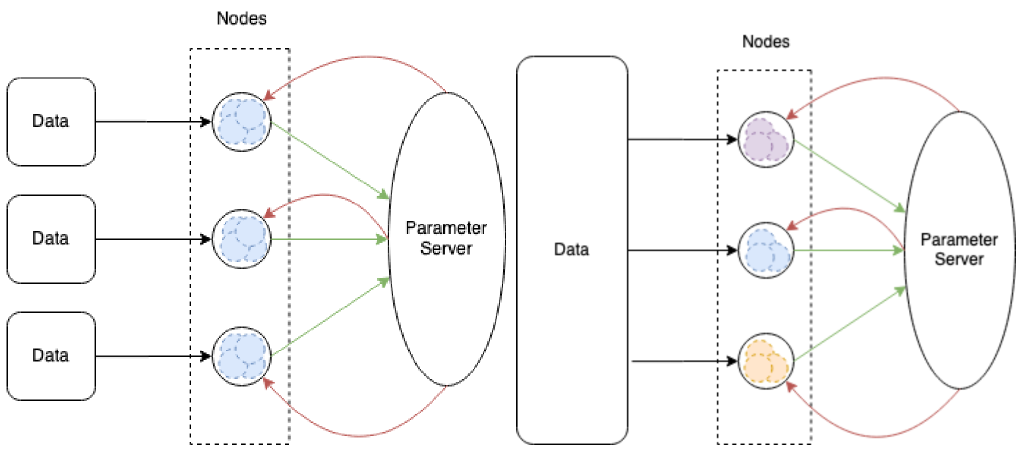


Figure 2: Left: Data Parallelism. Right: Model Parallelism. In both diagrams the green lines indicate local parameters being sent to the parameter server and red lines indicate parameters being sent to the worker nodes.

training data is passed through each node. While in Data Parallelism its common for each node to hold the whole machine learning model.

The key advantage of Model Parallelism is that machine learning models can be far larger as they no longer have to sit on one machine. However this one great advantage comes with some disadvantages. Some parameters may take more time to converge than others, this means that at times some nodes may be idle while others are still converging, so the spread of computation is not equal or efficient. [12] Because some parameters converge at different rates a scheduler can be used, which does improve model convergence. However this requires more computational overhead and communication and reduces iteration throughput. [11]

Data Parallelism has the benefit that data throughput can be very large, making processing using this method very fast. However with more nodes the communication overhead increases as the nodes must communicate the changes in their model parameters to each other. [20] The nodes can communicate with each other synchronously, but this means the computation is only as fast as the slowest node. If the nodes communicate with each other asynchronously then some of the calculations will be made on out of date model parameters so training examples may be needlessly wasted.

2.5 Low Power Hardware, IoT and Mobile Computation

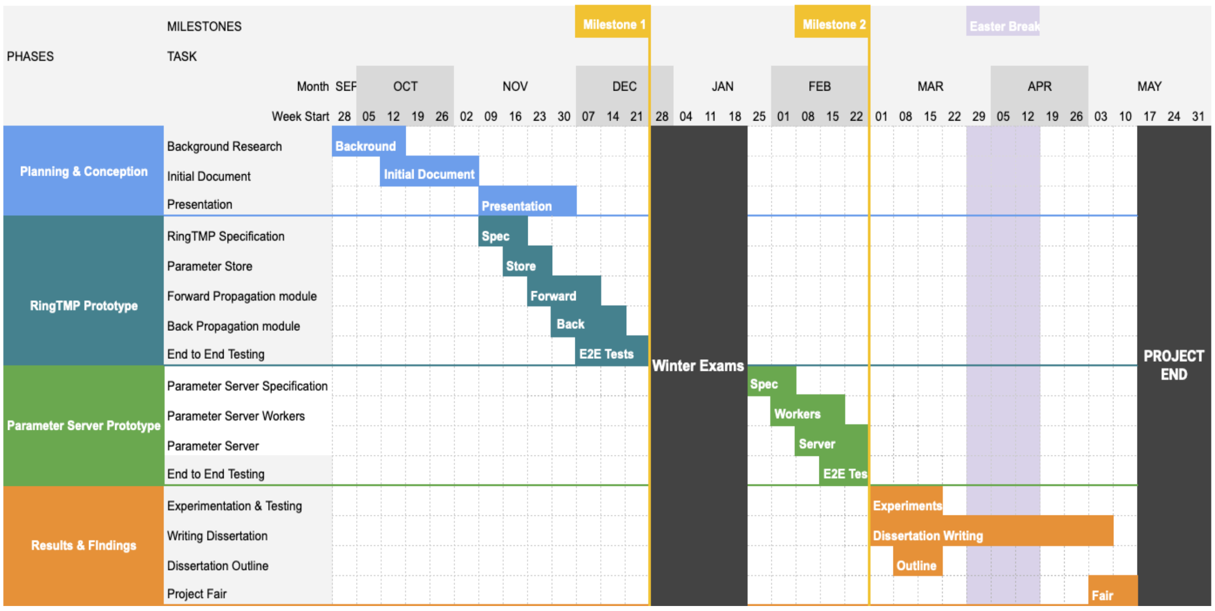
Historically Machine learning algorithms have been focused on high model accuracy through large models and vast amounts of training data, energy consumption and efficiency has rarely been taken into consideration. However with the rise of Internet of Things (IoT) devices and the established ubiquity of smart phones more data than ever is being produced. Soon this data generation with exceed the capacity of the internet, and experts estimate that over 90% of data will be stored and processed locally. [10] By extension this means machine learning algorithms will have to be performed locally too. This introduces some challenging issues. Modern machine learning algorithms require vast computational power and large amounts of data. Local devices don't have the capacity to hold large data sets or the power to compute large machine learning models in a viable amount of time, while many of them are also battery powered so power consumption becomes another issue.

A solution to this is to massively distribute the model over multiple decentralised nodes. The level of distribution is even greater than that of centralised compute clusters. In this solution each device computes a model using its own local data, infrequently (due to network constraints) the model is shared with a coordination server, which will then distribute the changes across all nodes in the network. [9] While this method is inefficient as the infrequent communications mean that many nodes may do much of the same work, and the merging of local models into a global one infrequently may cause loss of information. It still produces a model which converges in a relatively few rounds of communication. [8]

Efforts have been made in techniques to reduce the power memory and storage needed for machine learning algorithms to operate. One of these is Data stream mining. The idea is that a device can stream the analytics data directly into the model rather than storing the data in storage for later use. [21] This means after the data has been read by the model it is lost. But that also means that no data needs to be stored, meaning resources are not spent reading and writing to storage. This has an application in mobile devices, as they produce data at a low rate through user interaction. Therefore the model can be build in real time as actions occur. The data produced on the mobile itself may not be enough to effectively train the model, but via communication with other users distributed over the network the model can converge. [8]

From the research available there seems to be research into distributed computing on local devices, investigation in to power measurements of machine learning algorithms [8, 9] and power reduction of machine learning algorithms on local devices. [21] But there is a distinct lack of research into efficiency of *distributed algorithms* from the perspective of efficiency and power consumption on local devices. Having a more power efficient distributed machine learning algorithm, even if the optimisation was marginal on each device, would have an enormous effect on the output of the system, as many devices are connected.

3 Project Plan and Time Management



The gant chart above outlines the order and time frame each task should take in order complete this dissertation on time and to a high quality. Building a RingTMP prototype of my system is the most important task to embark on. It is better to do this first as it will take the longest time to produce and has the most 'unknown unknowns', moreover this is the centrepiece of my project, if I have not finished this there will be no dissertation to write. I then take a month break to revise for my exams. I acknowledge that this is a long time, however the semester 1 modules contribute more to my final grade than my dissertation does, therefore its important this project isn't detrimental to my grades in other modules. After Christmas I'll start work on a basic prototype of a parameter server. This will use the same tools and languages as my framework. In this way it will be easy to compare and contrast the benefits and shortcomings of each system on a level playing field. Once Both pieces of software have been complete I will run various tests to see how well my aims have been achieved. While I'm conducting these tests I shall also be writing up my results. Once my experiments have finished I shall start working on my dissertation document in earnest. I'll hand in my outline a week before easter break and shall keep working till it is complete.

3.1 Software Life Cycle

My methodology I will use to produce my software will be a variation of Agile methodology. Generally speaking Agile software development models tend to take the form of doing a small amount of specification upfront and thereafter working in small cycles to design, develop, validate and deploy a new version of software. Agile development places particular focus on showing customers working versions of the software as early as possible an being able to adapt to changing customer requirements easily. [22] However the pure Agile methodology does not completely suit my needs, as Agile programming is Customer focused and my project is not. Agile programming also focuses on changing customer requirements. However there will be no changing requirements from a customer as there is none. On the other hand optimising for working software over small iterations is a good philosophy for my work. Therefore my methodology will specify work upfront and then follow an agile workflow with week long iterations. I can do this as I won't be beholden to a customer. This will mean that I have more of an architectural design roadmap that if I were following a pure Agile methodology.

I also intend to take this iterative methodology to a smaller scale using Test Driven Development. [23] This is a philosophy where you write you tests before you write your code for each logical block. This means you specify the code's function before you have written it and you have solid acceptance criteria for when that piece of code is working. If a piece of code is too complex to test then that is an indication that your logical block needs to be broken down into small pieces. This way you end up with highly cohesive but loosely coupled components, and a high degree of confidence that they work. When introducing new features that effect existing code you can instantly see where it is breaking as the tests will fail, saving you valuable debugging time.

3.2 Research Methodology

My project lends itself to using an experimental research methodology to collect my findings, with this in mind is paramount to ensure that my results are taken is as controlled conditions as possible. I will achieve this by creating two software prototypes, one is my new novel machine learning framework, the other is the established parameter server design. I will make these using the same languages and tools as each other. Only the architecture will differ, meaning comparisons of the two systems will only reflect the performance of the respective architectures. On these systems I will conduct a series of tests. Each of these tests will compare each system to one another and will correspond to the aims I outlined in the introduction:

- To compare the efficiency of the systems we can measure the idle time of each worker in each of the systems and divide that by the time each system took to train a model on some basic task. This will give us a percentage of how much time the machine spend processing and how much time was spent on communication between nodes. This same experiment can be repeated with a different amount of nodes to see how the results change.
- To compare the progress made per iteration, the loss function can be measured for each iteration in both of the systems, then it is trivial to see which ones makes the most progress.
- To measure scalability both systems can be run with a varying number of different nodes. In all experiments they are given the same dataset to be trained on. We can measure how long it takes for them to complete the task, and how well both systems scale when more nodes are added.
- To measure resilience a node can be permanently or temporarily removed from each system while a model is being trained. Then the success of its mitigation and recovery strategies can be assessed.
- To discover the limits of how large a Neural Network each system can hold. Tests which incrementally increase the amount of layers and number of layers until the system crashes can be run.

By doing all this I can measure the performance between these two systems and accurately assess my architecture's performance relative to the parameter server.

3.3 Risk Assessment

A dissertation is a long arduous process in which many things could go wrong. Hence it is best to identify possible hazards and decide how to mitigate them before they happen, whilst also having a contingency plan should anything occur.

Risk: The RingTMP prototype is not finished due to time constraints.
Likelihood: Low
Potential Impact: Severe Impact on the grade for my dissertation.
Mitigation: I have thoroughly researched the tools and algorithms I need to complete this prototype and am familiar with them. This is the first task I will start work on. Meaning that I have the most time to complete it.
Contingency Plan: If I need more time I can reduce the scope of tasks further down the project timeline.

Risk: Winter exams interfere with my project timeline.
Likelihood: Low
Potential Impact: The project becomes behind schedule
Mitigation: The Winter exams have been factored into the project timeline, ensuring that I have enough time to revise and keep the project on schedule.

Risk: Parameter Server more complex than initially believed.
Likelihood: Medium
Potential Impact: The Project becomes behind schedule
Mitigation: I have read many papers on the operations of the parameter server and by being aware with its inner workings. Open source versions of the parameter server can be found that I can use as a template for my own.
Contingency Plan: If the task of creating the parameter server was greatly underestimated, then an open source version could be used though this would have repercussions for the meaningfulness of my findings.

Risk: I contract Coronavirus
Likelihood: Medium
Potential Impact: I have to rest for a number of days, and possibly have lasting symptoms for months after. [24]
Mitigation: By not returning physically to university, I reduce the probability of catching Coronavirus, as the cases are much lower in my current location. My physical social contact will also be kept to a minimum and good hygiene will be observed
Contingency Plan: I will have to isolate for 10 days after displaying symptoms. For the majority of tasks can be completed in isolation. If I am still exhibiting symptoms such as fatigue, cognitive impairment and breathlessness I may have to reassess the scope of the

project and apply for extenuating circumstances.

References

[1] S. M. McKinney, M. Sieniek, V. Godbole, J. Godwin, N. Antropova, H. Ashrafian, T. Back, M. Chesus, G. S. Corrado, A. Darzi, M. Etemadi, F. Garcia-Vicente, F. J. Gilbert, M. Halling-Brown, D. Hassabis, S. Jansen, A. Karthikesalingam, C. J. Kelly, D. King, J. R. Ledsam, D. Melnick, H. Mostofi, L. Peng, J. J. Reicher, B. Romera-Paredes, R. Sidebottom, M. Suleyman, D. Tse, K. C. Young, J. De Fauw, and S. Shetty, "International evaluation of an ai system for breast cancer screening," *Nature*, vol. 577, pp. 89–94, Jan 2020.

[2] A. Piergiovanni and M. S. Ryoo, "Avid dataset: Anonymized videos from diverse countries," 2020.

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, p. 107–113, Jan. 2008.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[5] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," 2020.

[6] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, (Broomfield, CO), pp. 583–598, USENIX Association, Oct. 2014.

[7] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," 2018.

[8] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[9] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine

learning," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 63–71, 2018.

[10] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[11] J. K. Kim, Q. Ho, S. Lee, X. Zheng, W. Dai, G. A. Gibson, and E. P. Xing, "Strads: A distributed framework for scheduled model parallel machine learning," in *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[12] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1223–1231, Curran Associates, Inc., 2012.

[13] E. Alpaydin, *Introduction to machine learning.* MIT press, 2020.

[14] C. Ma, J. Konečný, M. Jaggi, V. Smith, M. I. Jordan, P. Richtárik, and M. Takáč, "Distributed optimization with arbitrary local solvers," *Optimization Methods and Software*, vol. 32, no. 4, pp. 813–848, 2017.

[15] D. Newman, P. Smyth, M. Welling, and A. U. Asuncion, "Distributed inference for latent dirichlet allocation," in *Advances in neural information processing systems*, pp. 1081–1088, 2008.

[16] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proc. VLDB Endow.*, vol. 3, p. 703–710, Sept. 2010.

[17] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola, "Scalable inference in latent variable models," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, (New York, NY, USA), p. 123–132, Association for Computing Machinery, 2012.

[18] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, pp. 19–27, 2014.

[19] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, “Petuum: A new platform for distributed machine learning on big data,” *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[20] A. Elgabli, J. Park, A. S. Bedi, M. Bennis, and V. Aggarwal, “Gadmm: Fast and communication efficient framework for distributed machine learning.,” *Journal of Machine Learning Research*, vol. 21, no. 76, pp. 1–39, 2020.

[21] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, “Estimation of energy consumption in machine learning,” *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75 – 88, 2019.

[22] M. Fowler, J. Highsmith, *et al.*, “The agile manifesto,” *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[23] K. Beck, *Test-driven development: by example.* Addison-Wesley Professional, 2003.

[24] C. H. Sudre, B. Murray, T. Varsavsky, M. S. Graham, R. S. Penfold, R. C. Bowyer, J. C. Pujol, K. Klaser, M. Antonelli, L. S. Canas, E. Molteni, M. Modat, M. J. Cardoso, A. May, S. Ganesh, R. Davies, L. H. Nguyen, D. A. Drew, C. M. Astley, A. D. Joshi, J. Merino, N. Tsereteli, T. Fall, M. F. Gomez, E. L. Duncan, C. Menni, F. M. Williams, P. W. Franks, A. T. Chan, J. Wolf, S. Ourselin, T. Spector, and C. J. Steves, “Attributes and predictors of long-covid: analysis of covid cases and their symptoms collected by the covid symptoms study app,” *medRxiv*, 2020.