Doctoral Dissertation

An Infrastructure for Collaborative Machine Learning on Resource-Constrained Heterogeneous Environments

Kundjanasith Thonglek

Program of Information Science and Engineering
Graduate School of Science and Technology
Nara Institute of Science and Technology

Supervisor: Professor Hajimu Iida
Software Design and Analysis Laboratory (Division of Information Science)

Submitted on June 1, 2023

A Doctoral Dissertation
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Kundjanasith Thonglek

Thesis Committee:

Supervisor Hajimu Iida
(Professor, Division of Information Science)
Kazutoshi Fujikawa
(Professor, Division of Information Science)
Kohei Ichikawa
(Associate Professor, Division of Information Science)
Keichi Takahashi
(Assistant Professor, Tohoku University)
Chawanat Nakasan
(Lecturer, Kasetsart University)

An Infrastructure for Collaborative Machine Learning on Resource-Constrained Heterogeneous Environments*

Kundjanasith Thonglek

Abstract

Collaboration has been vital for the rapid and successful growth of the software industry. Software development infrastructures, such as GitHub for source codes and DockHub for container images, allow individuals from diverse backgrounds and organizations to work together and create complex and large-scale software that even big tech companies find it challenging to develop and maintain. However, such collaborative infrastructure is not yet available for machine learning models. This presents an opportunity to introduce LiberatAI into computer science for removing the barrier to the collaborative development of machine learning models from the limitation of data privacy and existing resource constraints.

In this dissertation, I propose LiberatAI, an infrastructure for collaboratively developing machine learning models that allow researchers to work together and potentially build better models than big companies can. LiberatAI applies federated learning to train the models while preserving data privacy. LiberatAI allows individuals to collaboratively train models on their environments, which are usually heterogeneous. Three modules in LiberatAI support training a model on diverse storage, computing, and communication resources. (1) Compressor module is proposed to reduce the model size to fit the storage capacity of the environment. (2) Aggregator module is proposed to aggregate the models trained

*Doctoral Dissertation, Graduate School of Science and Technology, Nara Institute of Science and Technology, June 1, 2023.

on heterogeneous computing resources. (3) Sparsifier module is proposed to sparsify the model for exchanging the model between a server and clients. LiberatAI was evaluated using state-of-the-art neural network models to detect COVID-19 cases from chest X-ray images. COVID-19 detection is one of the most popular machine learning applications for privacy-sensitive data. As a result, the ensemble model with heterogeneous structures on six different hardware environments from LiberatAI produces accuracy higher than a trained single COVID-NET by 5.39%.

Keywords:

Collaborative Development, Distributed Computing, Edge Machine Learning, Federated Learning, Privacy Preservation, Resource Heterogeneity

Contents

List of Figures

List of Tables

1. Introduction

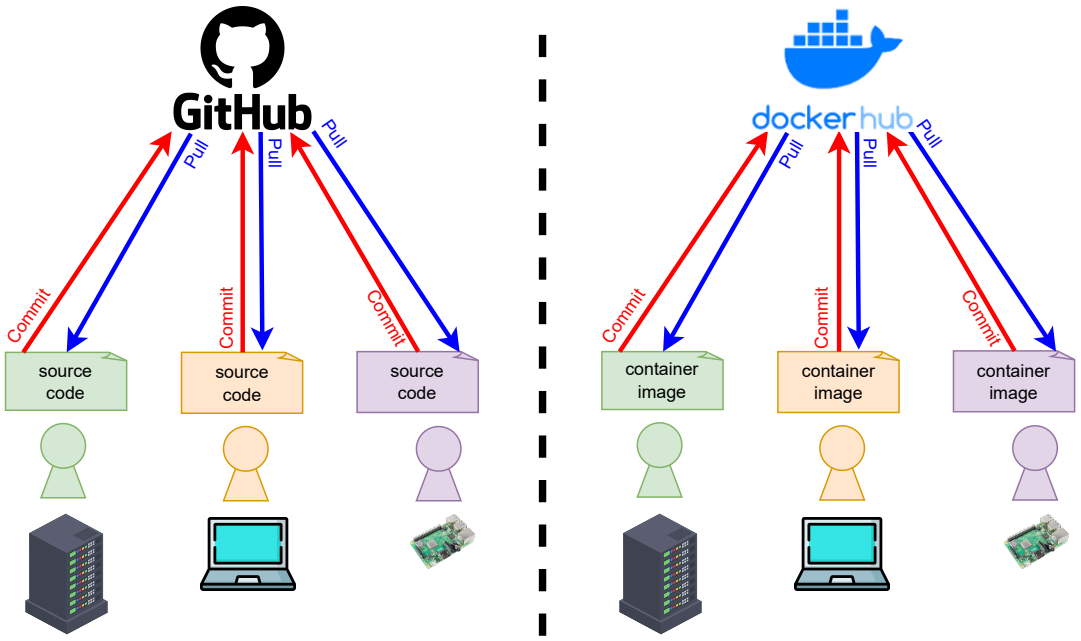


Figure 1: Collaboration on GitHub and DockerHub

Collaboration is crucial for fostering creativity, driving innovation, and achieving success in our increasingly interconnected society. Collaboration has been vital for the rapid and successful growth of the software industry. Software development infrastructures, such as GitHub for source codes and DockHub for container images as shown in Fig. 1, allow individuals from diverse backgrounds and organizations to work together and create complex and large-scale software that even big information technology companies find it challenging to develop and maintain. However, such collaborative infrastructure is not yet available for machine learning models. This fact has led to the current situation where state-of-the-art machine learning models can only be built by a handful of big companies such as Google, Amazon, Meta, and Microsoft, which have access to gigantic datasets and massive amounts of computing resources. Inspired by the success of software development infrastructures, an infrastructure for collaboratively developing machine learning models could allow researchers to work together and potentially build better models than big companies can.

The simplest approach for building a machine learning model by collective effort is to aggregate the datasets contributed by participants into a single repository and train a model using the aggregated datasets. However, this approach is often infeasible with privacy-sensitive data, because data privacy, security, and ownership prevent institutions from sharing their data with others. As a result, machine learning models built through collaboration were trained only using public datasets, and their size and diversity were inherently limited.

Federated learning was proposed to preserve data privacy for building a machine learning model collaboratively. In federated learning, each client trains the model over their local dataset on their environments and then each client uploads the trained model to a server without exposing the training dataset. Afterward, the trained models from every client are aggregated on a server. However, training the same model on all clients is infeasible due to resource constraints. For this reason, I proposed an infrastructure to handle training the model on heterogeneous storage, computing, and network resources.

1.1 Motivation and Goal

Building a highly accurate machine learning model requires a well-designed model architecture and a high-quality training dataset [1]. Since it is challenging for a single researcher to build a well-designed model and high-quality dataset, collaborative development to exchange valuable knowledge and experience among domain experts is imperative [2]. Such collaboration has been vital for the rapid and successful growth of the software industry [3]. Online software development platforms such as GitHub allow individuals from diverse backgrounds and organizations to work together and create complex and large-scale software [4]. However, software development platforms are designed to manage source codes and are not suitable for building datasets.

Datasets for machine learning are usually constructed from data collected from many data sources to increase their size and diversity as shown in Fig 2. However, sharing data with each other is not always possible due to privacy policies and license limitations [5]. Therefore, federated learning has been proposed to train a machine learning model while preserving data privacy [6]. In federated learning, the data remains on the devices or servers where it is collected, and only model

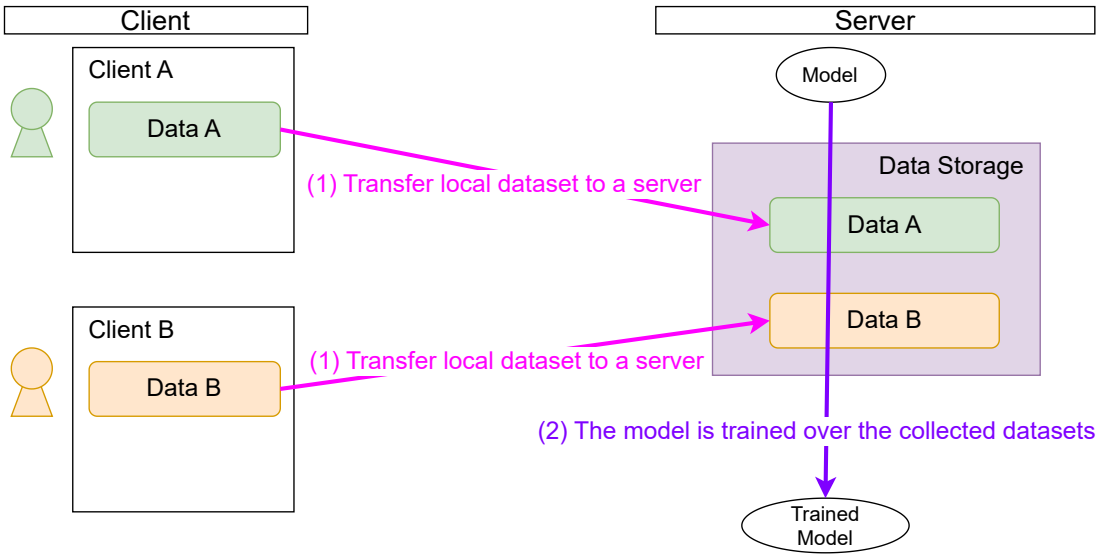


Figure 2: Overview of traditional collaborative machine learning

updates are exchanged between the parties involved in the training process as shown in Fig 3. This helps to address privacy concerns. It also enables model training on a much larger and more diverse dataset, as the data can come from multiple sources.

In practical situations, however, it is not always possible for all user devices to use the same model due to the resource constraints of edge devices that limit the ability to purchase high-performance computing and storage systems. Additionally, the available physical space, power supply, and network quality are also limiting factors. Hence, enabling federated learning in heterogeneous environments is important in achieving high model accuracy since the large size and diversity of the training dataset are necessary to train the model from diverse environments. The limitation of each heterogeneous client environment is considered when the machine learning models are trained or used for inference [7].

Since each model requires different hardware resources for training or inference, I focus on the heterogeneity of existing client hardware resources. In this dissertation, I consider three aspects of hardware heterogeneity: storage, computing, and communication. To the best of my knowledge, a federated learning platform that considers the various limitations of heterogeneous environments on clients does not exist yet.

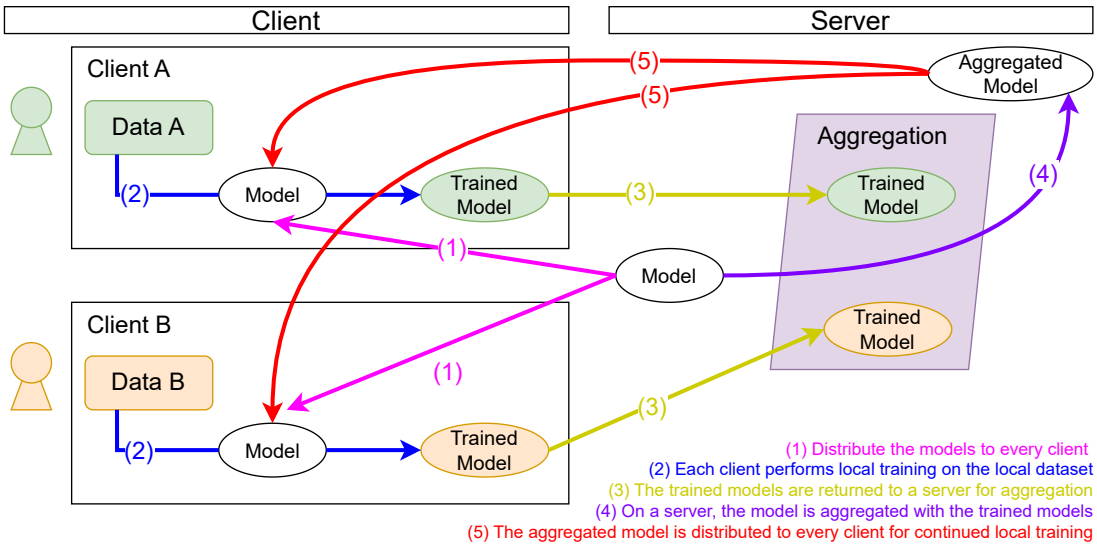


Figure 3: Overview of federated learning

This dissertation proposes *LiberatAI*, an infrastructure that enables the collaborative development of machine learning models on heterogeneous environments while preserving data privacy. Figure 4 shows the overview of LiberatAI. Federated learning is applied to train the models without exchanging the raw dataset between a server and clients. LiberatAI extends my three previous works to support training a model on diverse storage, computing, and communication resources. First, LiberatAI employs my method for reducing the model size to fit in heterogeneous storage capacity constraints [8]. Second, LiberatAI employs my method for aggregating the heterogeneous trained models from diverse computing resources [9]. Third, LiberatAI employs my method for sparsifying the models for saving the communication cost when the models are exchanged between a server and the clients [10]. The contribution of this dissertation is to build a collaborative machine learning infrastructure that integrates these my previous technologies to allow users to use appropriate models and reduce the required storage, and communication cost for each client so that it accommodates a variety of devices in edge computing environments. As a result, I show how LiberatAI efficiently builds models across heterogeneous environments.

I expect that LiberatAI will remove the barrier for the collaborative develop-

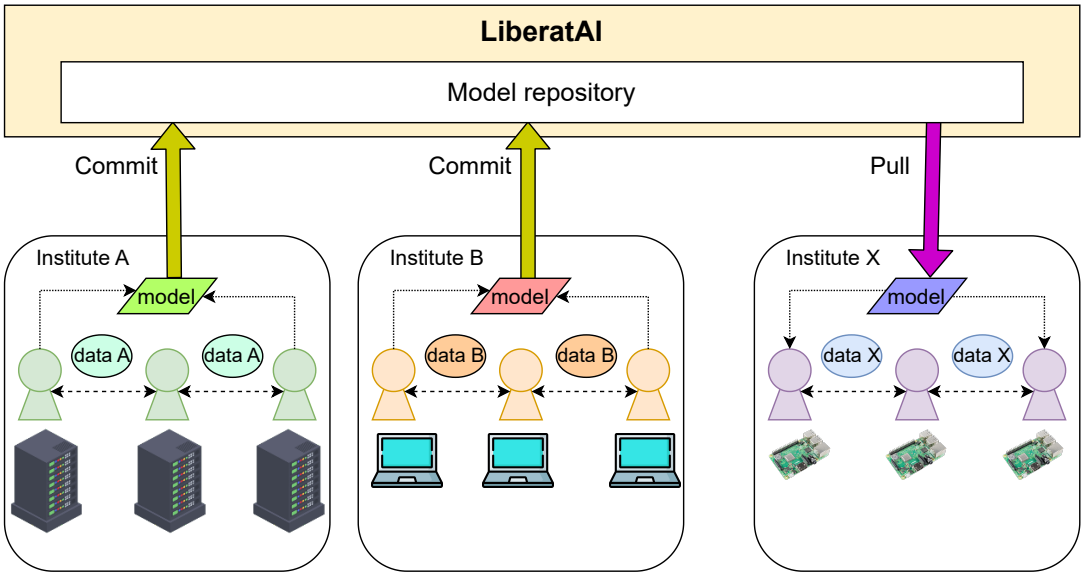


Figure 4: Overview of LiberatAI

ment of machine learning models from the limitation of data privacy and heterogeneous environments. It is often infeasible to distribute the same model to every edge device because of hardware limitations such as computing performance and storage space. Traditional collaborative development of machine learning models requires sharing their dataset and retraining the model from scratch. Many machine learning models will be built to support multidisciplinary research since researchers are able to contribute the existing models without training the models from scratch which requires a significant amount of computing resources.

LiberatAI will allow machine learning developers or researchers to develop machine learning models collaboratively. Research communities in both academia and industry will be expanded and crossed over multidisciplinary because of the infrastructure. The number of research collaborations will be continuously increased because the barrier of data usage and hardware resources has already been eliminated. LiberatAI might enable emerging models in various research fields especially the fields that utilize privacy-sensitive data when it is available. Furthermore, LiberatAI will attract many researchers to build research communities by sharing their knowledge and experience with each other.

1.2 Organization of the Dissertation

The rest of this dissertation is structured as follows. Chapter 2 explains the proposed method for reducing the size of models to fit in heterogeneous storage resources while maintaining the accuracy of models. This chapter evaluates the proposed method using the compression of various neural network models for classification, regression, and semantic similarity tasks. Chapter 3 describes the proposed method to aggregate the diverse models trained on heterogeneous computing resources. This chapter evaluates the proposed method to ensemble the models with four different structures for image classification. The optimized weights of weighted average ensembling are demonstrated to weight each model structure. Chapter 4 shows the proposed method to sparsify the models for exchanging the models between a server and clients on heterogeneous network resources. This chapter compares the proposed method to the existing methods on communication cost and model accuracy. Chapter 5 presents how I integrate my proposed methods to handle training the models on heterogeneous environments for building LiberatAI infrastructure. This chapter evaluates LiberatAI using state-of-the-art neural network models to detect COVID-19 cases from chest X-ray images, which is one of the most popular machine learning applications for privacy-sensitive data. Lastly, chapter 6 concludes this dissertation and discusses future works.

2. Training Models with Heterogeneous Storage Resources

2.1 Introduction

Edge Machine Learning is gaining much attention from the academia and industry because edge devices have more computing power than ever [11]. Figure 5 presents the workflow of Edge ML, where models are deployed and executed on edge devices instead of cloud servers. Edge ML provides better data privacy and less response time because datasets do not need to be uploaded to the cloud as in conventional cloud-based machine learning [12].

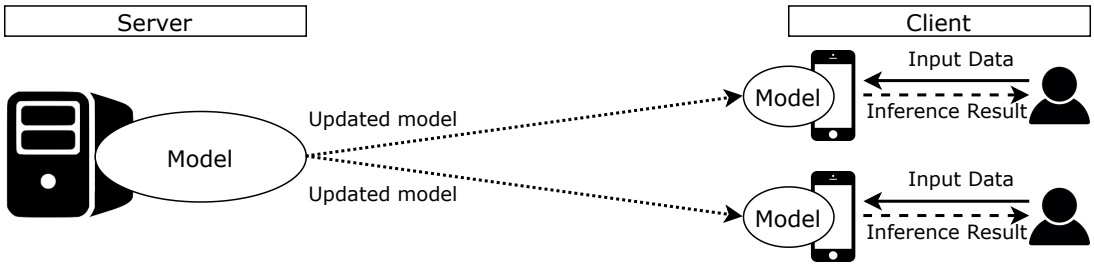


Figure 5: Workflow of Edge Machine Learning

The key restrictions for running machine learning models on edge devices are limited computing resources and storage space [13]. The amount of computing resources that is required to run a model depends on its size and complexity [14]. In particular, recent neural networks involve complex layer structures and many parameters, which requires a large amount of computing resources. Thus, researchers have recently studied various model compression methods to reduce the required storage space for deploying machine learning models on resource-constrained edge devices [15]. In this chapter, I particularly focus on reducing the model size for storing and transferring the models due to the limitation of storage space on edge devices. Compressing neural network models reduces the size of the model, but as a trade off, the compression loses accuracy. This is because compression might eliminate some of the weights that have a significant impact on the accuracy of models [16]. Minimizing the loss of model accuracy while maximizing the compression ratio is the challenge in model compression.

To minimize the accuracy loss of compressed models, finding an optimal configuration for the compression and retraining process are important. Several model compression techniques including low-rank factorization [17], optimal brain surgeon [18], learning structured sparsity [19] and temporal sequence modeling [20] have been proposed recently. However, these existing techniques require a significant amount of trial and error to find the optimal configurations for compressing the model size without significant accuracy loss [21].

Retraining is a method for increasing the accuracy of a trained model by performing the training process again. Previously proposed retraining methods usually rely on labeled datasets [22]. A labeled dataset is a collection of samples that have already been labeled to classify specific object features. In the training of neural network models, labeling is a crucial step [23], and is a lengthy process [24]. Labeling is especially expensive when it requires knowledge from domain experts. Moreover, labeled datasets are not always available due to license restrictions and privacy policies (e.g., patent licenses, confidentiality agreements, and general data protection regulation). For this reason, retraining with unlabeled datasets is incredibly useful when labeled datasets are inaccessible.

This chapter proposes a method to automatically find the optimal configurations for compressing and retraining the neural network models without labeled data. The proposed method employs a multi-objective optimization algorithm to simultaneously minimize model size and maximize model accuracy. The proposed method suggests multiple compression configurations that generate models with different model size and accuracy, from which users can select the configurations that suit their needs. Furthermore, I propose a retraining method that does not require the labeled dataset to be a training dataset. This chapter extends my previous work [8] on retraining quantized neural networks and removes the need for hand-tuning by users to find the best configuration. The contribution and novelty of this chapter are summarized as follows:

- I propose a method that automatically finds the optimal configurations for compressing and retraining models.
- I propose a retraining method for compressed models that does not require labeled datasets.

A motivating use case of the proposed method is to create multiple models tailored to the hardware constraints of diverse devices. For example, consider a voice recognition model for a voice assistant application that runs on mobile devices. Although a minor loss in accuracy is acceptable since there are no serious consequences even if the model output is incorrect, users would still want to run a model that achieves the highest accuracy within the hardware constraints of their mobile devices. In such case, finding a single compression configuration that fits all users is not possible. On the other hand, the proposed method can provide multiple configurations each with different hardware requirements.

2.2 Background

This section describes a brief overview of existing techniques for compressing and retraining neural network models.

2.2.1 Compression of Neural Networks

Recent neural networks are becoming increasingly larger to achieve higher accuracy [25]. However, not every parameter in a model contributes to the model accuracy. Based on this observation, model compression eliminates unnecessary parameters in a model with an aim to reduce its size while maintaining its original accuracy. Designing an efficient model compression method requires understanding the structure of a model, identifying redundant parameters, and eliminating those parameters with acceptable loss of accuracy.

Cheng et al. [26] surveyed various methods for compressing neural networks. They are classified into four approaches: (1) transferred/compact convolutional filters, which redesigns a compact model using convolutional filters with reduced size [27], (2) knowledge distillation, which redesigns a compact model of the entire model [28], (3) low-rank factorization, which reduces the model size using matrix decomposition [29], and (4) parameter pruning and sharing, which eliminates redundant parameters [30]. Table 1 shows the comparison of these existing approaches.

Transferred/compact convolutional filters and knowledge distillation redesign a compact model with a new structure to reduce the model size. Therefore, these

Table 1: Approaches for model compression

Approach	**Uses pre-trained models**	**Supports fully connected layers**	**Reduces redundant parameters**	**Loss of accuracy**
Transferred/compact convolutional filters	✗	✗	✗	Small
Knowledge distillation	✗	✓	✓	Large
Low-rank factorization	✓	✓	✗	Large
Parameter pruning and sharing	✓	✓	✓	Small

approaches require extra effort including redesigning a new model architecture and retraining the model compared to simply reusing pre-trained models. On the other hand, low-rank factorization and parameter pruning and sharing preserve the original structures of the models, and thus can efficiently reuse the results of the pre-trained models. Low-rank factorization reduces the model size by using matrix decomposition, but the decomposition of large matrices is computationally intensive, and the accuracy of the obtained model is generally low. Parameter pruning and sharing is a simple approach that reduces only the unnecessary and redundant parameters that have little impact on accuracy. It does not require much computation and the loss of accuracy is minimal.

Frankle et al. [31] proposed the lottery ticket hypothesis, which states that a dense neural network contains a subnetwork that can achieve the same level of accuracy as the original network after training. Based on this hypothesis, they proposed a pruning method for reducing the size of neural network. Diffenderfer et al. [32]. also developed a method based on the lottery ticket hypothesis that randomly splits a model into subnetworks and trains each subnetwork separately A subnetwork is then removed from the original model if the accuracy of the subnetwork is low. However, removing parameters from the model might degrade the model accuracy significantly because it changes the model architecture [33]. Since the parameter sharing approach does not affect the model architecture,

parameter sharing maintains better accuracy than parameter pruning.

Quantization is a widely adopted method based on the parameter sharing approach for compressing neural networks. It groups parameters into multiple clusters, and replaces all parameters in the same cluster with a representative value [34]. As a consequence, the size and computational cost of the model are reduced in return for slightly degraded accuracy. Vector quantization, one of quantization methods, outperforms other quantization approaches in compressing fully connected layers [35], which are known to be the most storage-demanding layers. Therefore, I applied vector quantization to compress the models.

2.2.2 Retraining Compressed Neural Networks

Applying retraining to compressed neural networks has been shown to increase the accuracy of compressed models. Retraining is a technique for improving the accuracy of an already trained neural network model by repeating the training process [36]. During the retraining process, the weights in a selected subset of layers are updated.

Sung et al. [37] demonstrated that highly complex models can absorb the effects of applying weight quantization by retraining, but neural networks with a limited number of connections cannot. They showed how the retraining method affects the resiliency of quantized networks.

Chen et al. [38] developed L-DNQ, a layer-wise quantization algorithm for neural networks that requires only a small subset of the original training dataset. They proved that the final quantization error of a neural network is bounded by a linear combination of the layer-wise quantization errors, and formulated quantization as a discrete optimization problem. A highly efficient algorithm called Alternative Direction Methods of Multipliers (ADMM) is used to find the solution for this optimization problem.

Early works showed the existing model compression and retraining methods require manual effort to find the optimal configurations. The configuration indicates which parameters should be quantized and how much they can be reduced without significant loss of model accuracy. Finding the optimal configurations manually wastes computational cost and cannot confirm that the configurations are optimized.

2.2.3 Automated Compression of Neural Networks

Recent methods for automatic compression of neural networks take advantage of deep reinforcement learning to efficiently find a compression configuration that reduces the model size while maintaining model accuracy. He et al. [39] proposed AMC to automatically search the configurations to prune a neural network model using deep reinforcement learning. AMC has two search protocols for resource-constrained and accuracy-guaranteed compression. Elthakeb et al. [40] developed ReLeQ, which is also based on deep reinforcement learning. It uses an asymmetric reward formulation to control the trade-off between accuracy and compression rate. Lou et al. proposed AutoQ [41] which is a two-level hierarchical deep reinforcement learning to automatically quantize the weights in the kernel-wise and the activation in the layer-wise.

All of these existing automated compression methods utilize single-objective optimization since they formulate a single reward function based on model size and model accuracy. Thus, these methods can only provide a single configuration, while my proposed method is based on multi-objective optimization and provides a set of optimal configurations.

2.3 Methodology

This section explains the proposed method to automatically find the optimal configurations for quantization and retraining neural network models without labeled data.

2.3.1 Overview

Figure 6 shows the four steps in my proposed method: (1) quantization, (2) retraining, (3) compression and (4) optimization. First, the original model is quantized using vector quantization. Second, the quantized model is retrained without labeled datasets to increase the model accuracy while maintaining the same model size. Third, the retrained model is compressed using gzip. Fourth, I measure the size and accuracy of the retrained model and then use a multi-objective optimization algorithm to find a set of potentially better quantization parameters. These steps are repeated until convergence is reached.

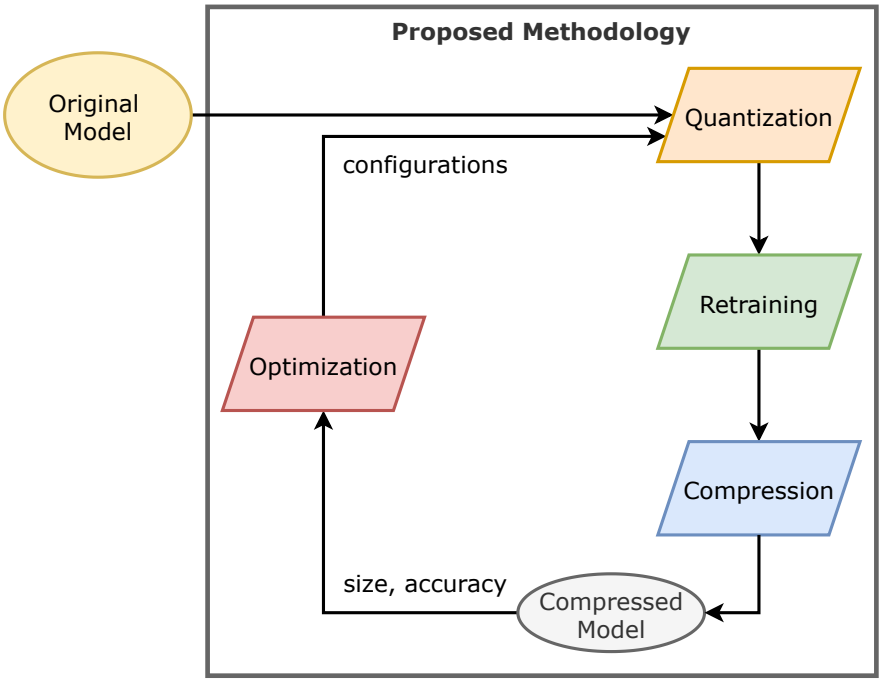


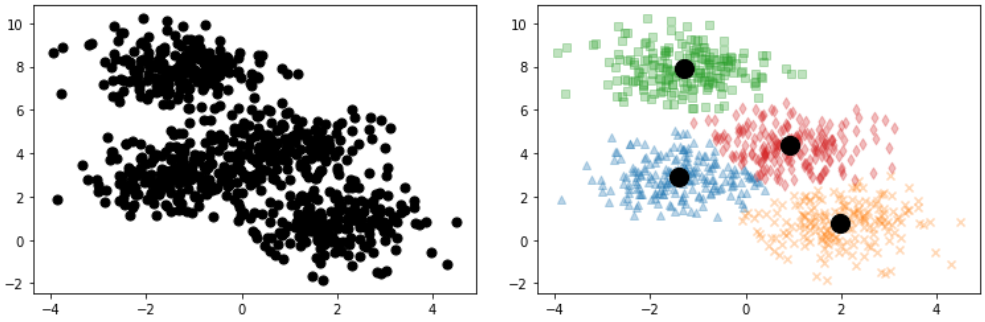
Figure 6: Overview of the proposed method

2.3.2 Quantization

Vector quantization is applied within a compressor module of LiberatAI, which is a technique used in data compression to divide a large space into smaller parts and represent the values within each part with a representative value, known as the centroid. When applied in a compressor module, vector quantization can help reduce the number of bits needed to represent the parameters of a neural network, leading to a smaller storage requirement.

For instance, the vector quantization is applied to quantized data into four clusters and centroids as shown in Fig. 7. Initially, the parameters are divided into four equally sized groups. The centroid, or representative value of each group, is then calculated. The parameters within each group are then represented by their corresponding centroid. As a result, data before and after vector quantization is shown in Fig. 7a and Fig. 7b, respectively.

In my method, I automatically tune the number of centroids using an optimization method described later in Section 2.3.5. Using few centroids reduces the



(a) Before vector quantization (b) After vector quantization

Figure 7: Data before and after vector quantization

model size, but degrades the model accuracy. Since the number and distribution of parameters vary across layers, the optimal number of centroids is different for each layer.

2.3.3 Retraining

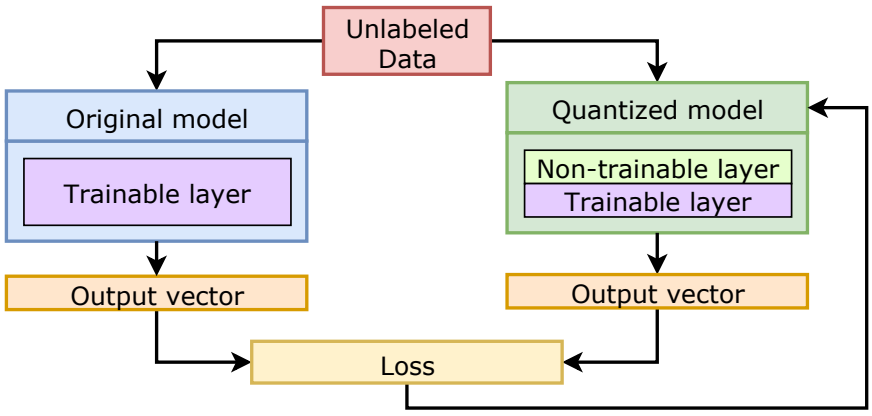


Figure 8: Retraining quantized model without labeled data

Following the quantization described in the previous section, I retrain the quantized model to recover the accuracy without using the original labeled dataset. Figure 8 illustrates how my retraining method functions. I first separate the layers in the target model into trainable and non-trainable layers. Specifically, the quantized layers are identified as non-trainable and the rest as trainable. The

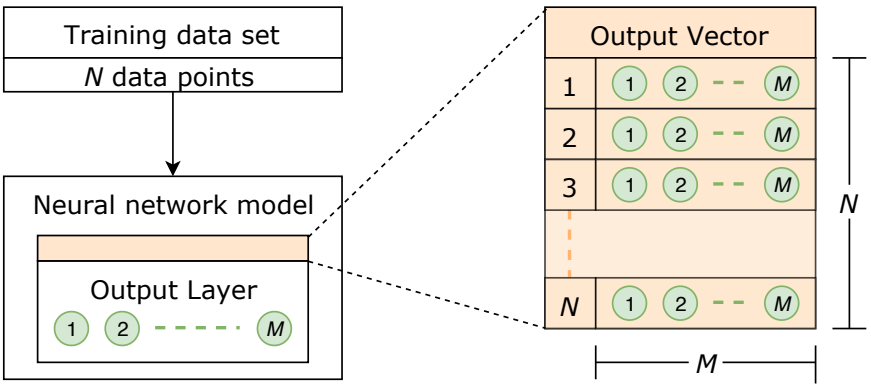


Figure 9: An output from a neural network model

quantized layers are not retrained because redundancies have already been eliminated and retraining would introduce redundancies again. Retraining only the non-quantized layers reduces the number of parameters that need to be updated during the training process. Accordingly, the proposed retraining method decreases the training time per epoch, and thus reduces the total retraining time as well.

I then retrain the quantized model by using the output from the original model as a teacher signal. To achieve this, both the original and quantized models receive the same unlabeled dataset, and the output vectors from both models are retrieved. Figure 9 illustrates an output from the last layer of a model, which represents the confidence for each class. The output is an $N \times M$ matrix, where M and N represent the number of output classes and samples, respectively. The quantized model is retrained to minimize the loss between the outputs from original and quantized models. Compared to using a labeled dataset, my method incurs additional overhead because outputs need to be generated from the original model. However, the runtime required for generating outputs using the original model is small compared to the runtime for retraining (approximately 2.8% of the retraining time).

2.3.4 Compression

The quantization step in my method increases the redundancy in a model by replacing the parameters with their representative values. However, the quanti-

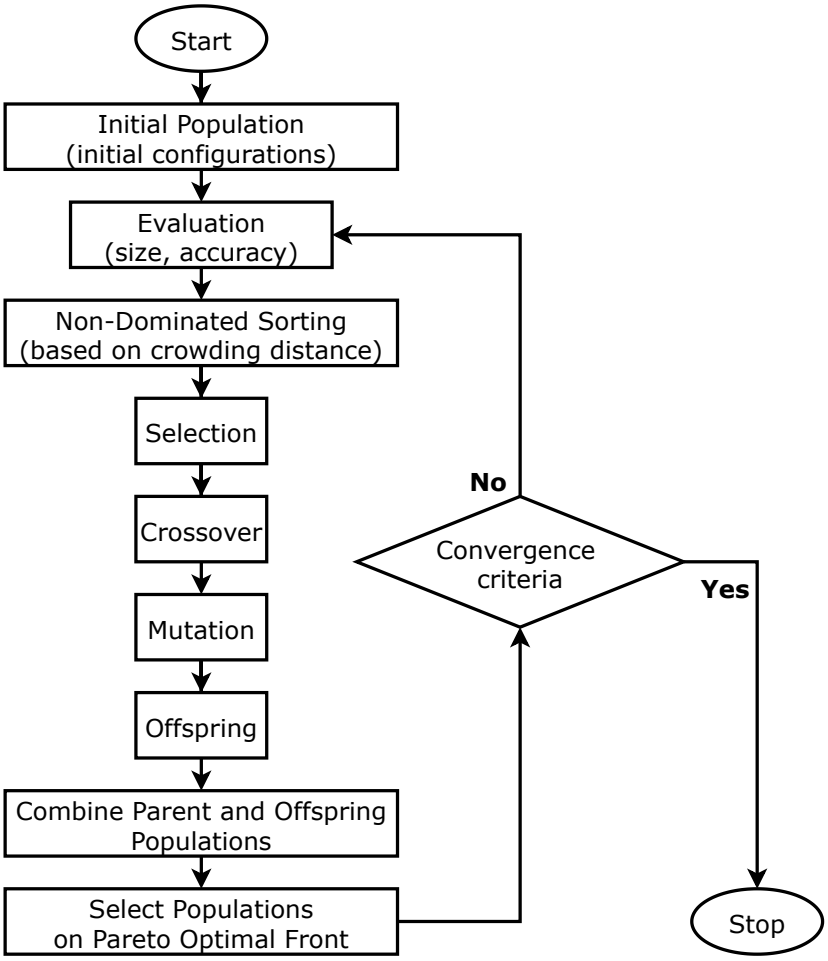


Figure 10: Flowchart of NSGA-II

zation step only makes a model amenable to compression and keeps the model size. In the last step of my method, I use a compression algorithm and eliminate the redundancy in a model to reduce its size.

In this work, the quantized model is saved as a Hierarchical Data Format 5 (HDF5)[1] file after retraining. Subsequently, gzip[2] is used to compress the model by finding the redundant parameters in the quantized layers. Gzip is one of the most efficient compressor and decompressor since it can compress almost any file type and is fast enough to compress and decompress data on the fly [42].

[1]http://www.hdfgroup.org/HDF5

[2]https://www.gnu.org/software/gzip/

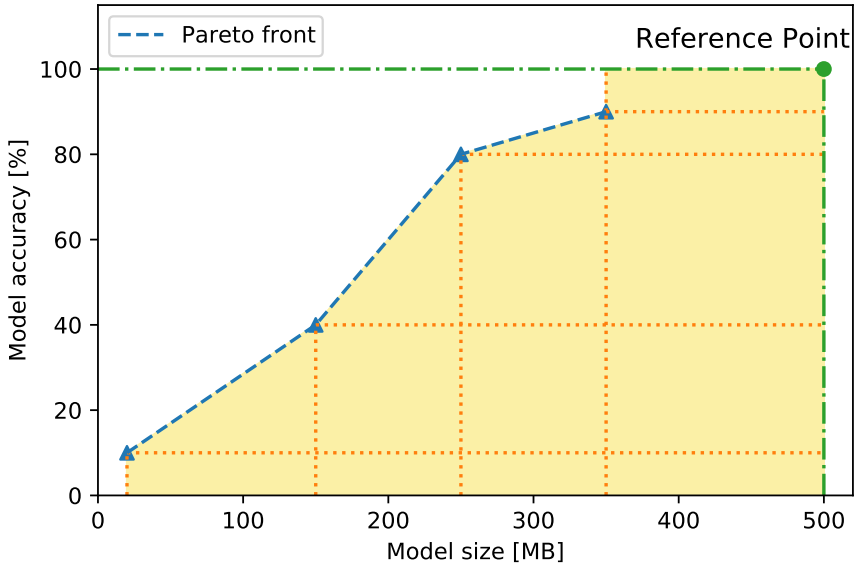


Figure 11: Calculation of hypervolume

2.3.5 Optimization

Finding the best configuration for quantizing a neural network model can be thought of as a multi-objective optimization problem because a configuration that maximizes the model accuracy and minimizes the model size at the same time needs to be found. Although considering other constraints such as memory footprint, inference time and convergence speed, are interesting, I focus on model size and accuracy only for simplicity in this chapter. Multi-objective optimization algorithms can find the optimal configurations faster than a full search. Thus, I apply the Non-dominated Sorting Genetic Algorithm II (NSGA-II), a state-of-the-art multi-objective optimization algorithm, to optimize the quantization configurations.

NSGA-II is a genetic algorithm for multi-objective optimization [43]. There are three distinctive features of NSGA-II: (1) it uses an elitist principle where elites of a population are given the opportunity to be carried to the next generation, (2) it applies crowding distance, which is an explicit diversity preserving mechanism, and (3) it emphasizes the non-dominated solutions. The optimization for model compression should focus on the non-dominated solutions since

smaller models might have the possibility to have higher accuracy than the larger one.

Figure 10 shows the workflow of NSGA-II. NSGA-II starts by generating an initial population (i.e., quantization configurations in the proposed method). Next, the optimizer evaluates the compressed models that applied the initial configurations to obtain their model size and accuracy. NSGA-II uses a non-dominated sorting method based on crowding distance to provide the solution as close to the Pareto front as possible. The quantization configurations are ordered by crowding distance in the objective space. Subsequently, it generates the offsprings from the previous generation using genetic operations including selection, crossover, and mutation. Next, the offspring configuration is applied to compress the model and then evaluate the compressed model. Lastly, select the optimal configurations on the Pareto optimal front are selected to calculate the convergence of the optimization process. These steps are repeated until the Pareto front has not changed for five generations.

I use the Hypervolume of the objective space to test the convergence [44]. Hypervolume calculates the volume between the Pareto front and the reference point as shown in Fig. 11. I defined the reference point *(the size of the original model, 1.0)* for both classification and regression because the highest possible accuracy and the R^2 score of the classification and regression model is 1.0.

2.4 Evaluation

Table 2: Specification of classification models

Name	**Size [MB]**	**Accuracy**
LeNet5	114.06	0.837
DenseNet201	80.28	0.728
ResNet152	235.64	0.748
VGG16	553.43	0.751
C3D	869.11	0.826

This section evaluates the proposed method to investigate the effectiveness of

Table 3: Specification of regression models

Model	**Size [MB]**	**R^2 score**
RNN	175.08	0.846
Bi-RNN	325.13	0.849
LSTM	700.20	0.883
Bi-LSTM	1300.30	0.884
GRU	525.25	0.882
Bi-GRU	975.40	0.883

Table 4: Specification of semantic similarity models

Name	**Size [MB]**	**Accuracy**
BERT-base	440.27	0.719
BERT-large	1368.31	0.767
BERT-xlarge	5089.06	0.802

the proposed method using various real-world neural network models for classification, regression and semantic similarity tasks. I first describe the experimental setup and visualize the objective space during optimization. I then assess the trade off between model accuracy and size. I also investigate the impact of the multi-objective optimization algorithms and retraining datasets on the performance of the compressed models, and the runtime required to quantize models. Finally, I compare the proposed method to state-of-the-art automated quantization methods.

2.4.1 Experimental Setup

The proposed method was evaluated using five classification models (Table 2), six regression models (Table 3) and three semantic similarity models (Table 4). I used five image classification models: LeNet5 [55], DenseNet201 [56], ResNet152 [57], VGG16 [58], and C3D [59]. I used three regression models: the recurrent neural network (RNN), the long short-term memory network (LSTM), and the gated

Table 5: Datasets used for training and retraining

Model	**Training dataset**	**Retraining dataset**
LeNet5	HARecognition [45]	1D MNIST [46]
DenseNet201	ImageNet [47]	CIFAR 100 [48]
ResNet152		
VGG16		
C3D	3D MNIST [49]	ObjectNet3D [50]
RNN	SP500 Stock [51]	Historical Bitcoin Market [52]
Bi-RNN		
LSTM		
Bi-LSTM		
GRU		
Bi-GRU		
BERT-base	SICK2014 [53]	SNLI2015 [54]
BERT-large		
BERT-xlarge		

recurrent unit network (GRU) [60]. In addition, I created a bidirectional version for each regression model. A bidirectional layer computes the input data in two directions, one from past to future and another from future to past [61]. This design helps the network to understand the future state. For the semantic similarity task, I used three variants of the Bidirectional Encoder Representations from Transformers (BERT) [62], which are BERT-base, BERT-large, and BERT-xlarge.

I prepared two datasets for each model as shown in Tab. 5. The initial model is trained and validated using the first dataset, and the compressed model is

Table 6: Hyperparameters for NSGA-II

Hyperparameter	**Value**
Size of the population per generation	10
Number of offspring	10
Crossover method	Uniform
Mutation method	Inversion

Table 7: Hardware specification

Hardware	**Specification**
CPU	Intel Xeon E5-2650 v2 ×2
Main Memory	256 GB
GPU	NVIDIA Tesla P100
GPU Memory	16 GB

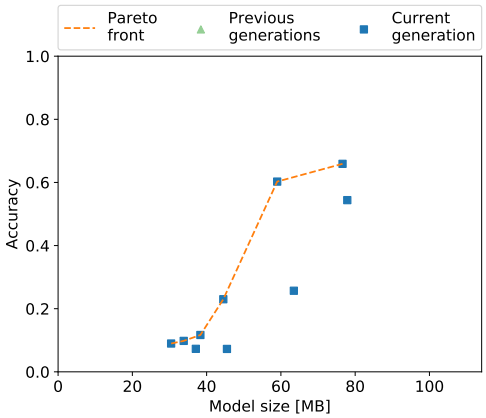
retrained using the second dataset. I used different datasets for the classification models because the dimensionality of the input they accept are different (LeNet5 accepts 1D, DenseNet201, ResNet152 and VGG16 accept 2D, and C3D accepts 3D inputs). The regression and semantic similarity models are all trained and retrained using the same datasets.

In this chapter, the number of centroids is chosen from powers of two ranging from one to 256. The used NSGA-II hyperparameters are shown in Tab. 6. Lastly, Tab. 7 presents the hardware used for the evaluation.

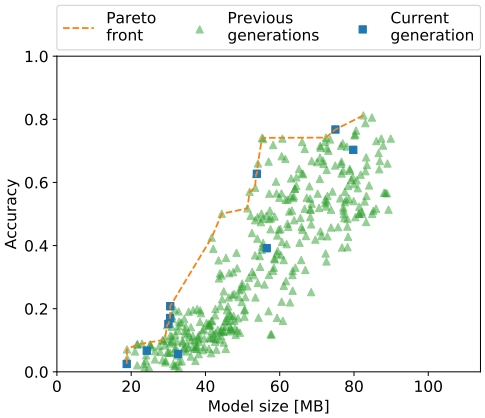
2.4.2 Visualization of the Optimization Process

To show the progress of the multi-objective optimization, I visualized the objective space along with the Pareto front at three points in time during the execution of the proposed method: (1) first generation, (2) half of the converged generation, and (3) the converged generation.

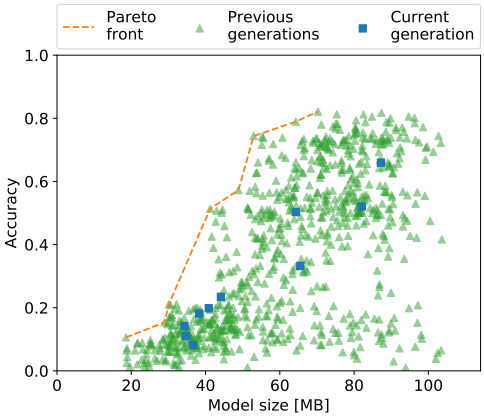
Figure 12 shows the objective space of LeNet5 when applying the proposed method from the first generation until the converged generation (53rd generation).



(a) First generation



(b) Half of converged generation

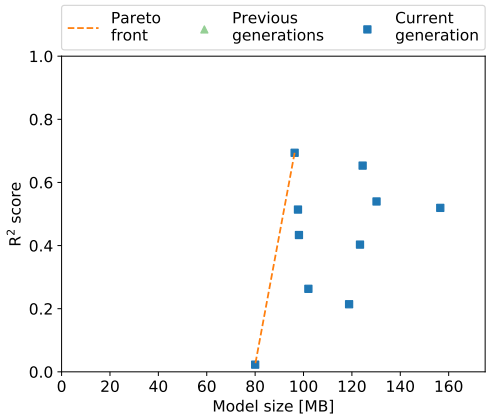


(c) Converged generation

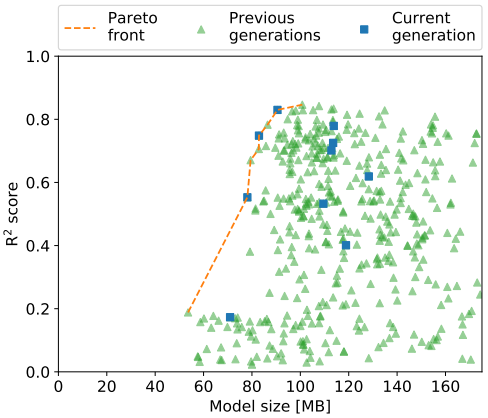
Figure 12: The objective space of LeNet5 when applying the proposed method

The hypervolume of the first generation is equal to 0.44 as shown in Fig. 12a. After half of the converged generation, the hypervolume is increased to 0.54 as shown in Fig. 12b. All optimal configurations keep changing through the first half of optimization. Figure 12c presents the objective space at the converged generation. For the last generation, the hypervolume is equal to 0.58. Also, there are eight configurations on the Pareto front which have a smaller size and a higher accuracy than the first half generation.

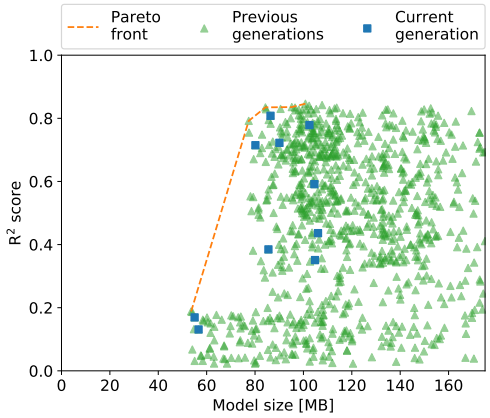
Figure 13 shows the objective space of RNN when applying the proposed method from the first generation until the converged generation. Figure 13a shows the objective space for the first generation which has hypervolume around



(a) First generation



(b) Half of converged generation



(c) Converged generation

Figure 13: The objective space of RNN when applying the proposed method

0.28. After half of the converged generation, the hypervolume is increased to 0.54 as shown in Fig. 13b. Figure 13c presents the objective space at the converged generation which has hypervolume around 0.57. I found out that the hypervolume grew when applying the proposed method. Increasing the hypervolume means the proposed method can find the better configurations when the number of generations rises.

The Pareto front moves towards a smaller model size and a higher model accuracy as generations pass. During the first half of the optimization, the Pareto front changes more quickly than during the second half of the optimization since the approximated Pareto front approaches the true Pareto front. For both classi-

fication and regression models, I observed that the optimal configuration on the Pareto front keeps changing when applying the proposed method.

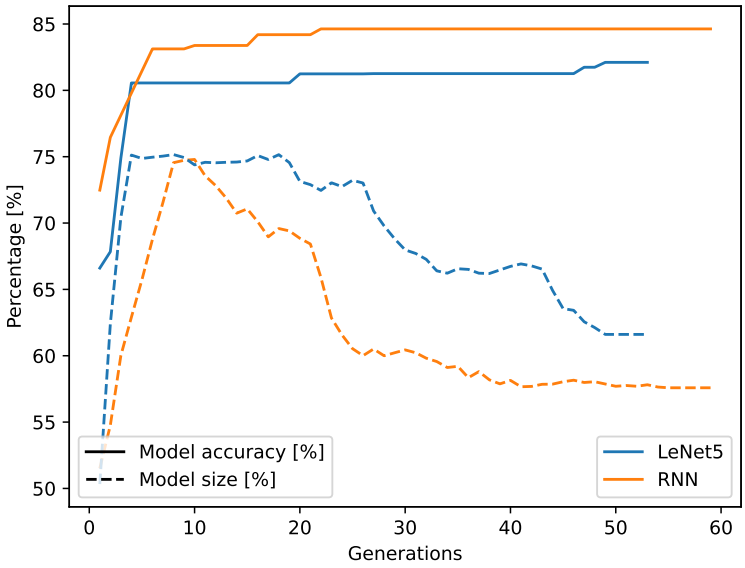
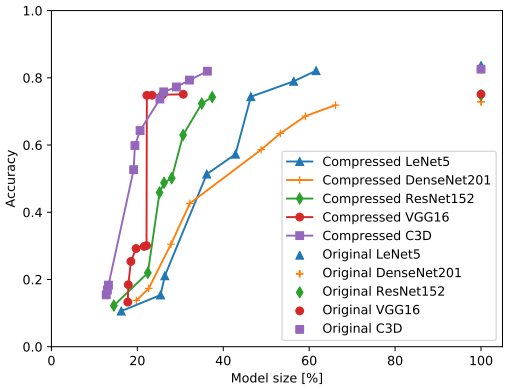


Figure 14: Percentage of model size and accuracy during optimization

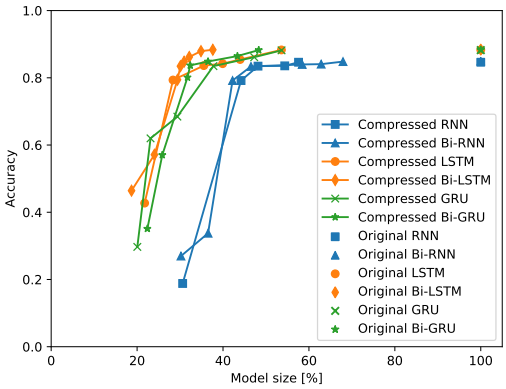
Furthermore, I analyzed the trade-off between accuracy, compression and computational cost by examining the optimal configuration on the Pareto front that achieves the highest accuracy for each generation. Figure 14 shows the model size and accuracy of LeNet5 and RNN during optimization. At the beginning of the optimization, the model size grew rapidly and reached a plateau after 5 to 10 generations in both models. In contrast, the decrease in model size was relatively slow during the optimization. In LeNet5, the model size converged after around 50 generations, while the model size of RNN took 40 generations to converge. If the optimization process is run to the end, a complete Pareto front will be found, but if a satisfactory configuration is found in the middle, the optimization process can be stopped at that point.

2.4.3 Model Size and Accuracy

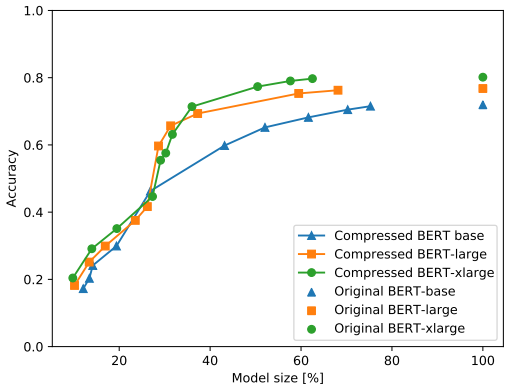
In this evaluation, I optimized each model using my proposed method until convergence and compared the Pareto fronts at the converged generation across models. Figure 15a shows the Pareto fronts for the five classification models. Here, the



(a) Pareto fronts for classification



(b) Pareto fronts for regression



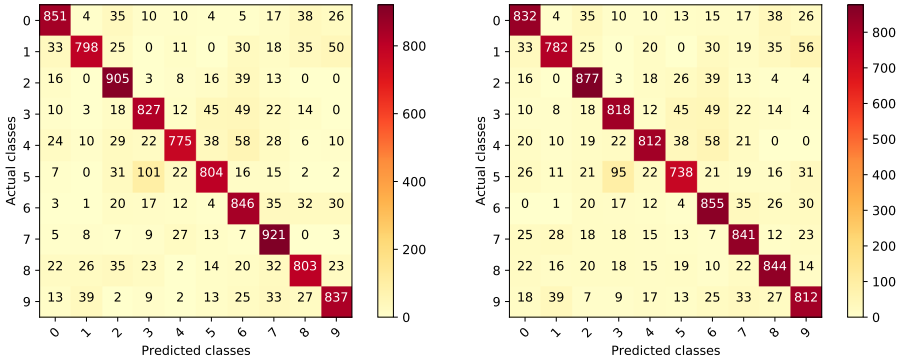
(c) Pareto fronts for semantic similarity

Figure 15: Pareto fronts

y-axis represents the accuracy while the x-axis represents the relative size compared to the original model. The figure indicates that larger classification models are more amenable to compression than smaller models since the Pareto fronts of the larger models are closer to the y-axis. Figure 15b presents the Pareto fronts for the six regression models. LSTM and GRU might be easier to compress than RNN because they contain more redundant parameters than RNN. Figure 15c shows the Pareto fronts for the three semantic similarity models. Larger BERT might be easier to compress than smaller BERT because larger one might contain more redundant parameters than smaller one.

What stands out in Fig. 15a is the sharp drop of accuracy under a certain model size. Taking LeNet5 as an example, its accuracy sharply declines when the

model size is reduced to less than approximately 46% of the original. The Pareto fronts for other classification models follow the same trend. The same trend is even clearer with the Pareto fronts for regression models shown in Fig. 15b. For example, RNN is compressed down to 40% of its original size with almost no degradation of accuracy, but steeply degrades when compressed smaller. I also observed the same trend with the Pareto fronts for semantic similarity models as shown in Fig. 15c.



(a) Original model (b) Compressed model

Figure 16: Confusion matrices of the original and compressed LeNet5 models

To assess the effectiveness of my compression method, I chose the models with the lowest accuracy loss on the Pareto fronts and compared them with their original models. The accuracy loss of all those models compared to the original models was less than 1%. For classification models, the model sizes of C3D and VGG16 are reduced to approximately 60% of their original size, whereas LeNet5 is reduced to 38.39%. This might stem from the fact that LeNet5 is the smallest model among the classification models, and the fraction of necessary parameters is larger than the models. For regression models, bidirectional models are reduced more than unidirectional models. This suggests that bidirectional models have more redundant parameters than unidirectional models. For semantic similarity models, larger models are compressed more than smaller models since larger one has more redundant parameters than smaller one. Finally, I visualized the confusion matrices for the original and compressed LeNet5 models as shown in Fig. 16. As expected, the confusion matrices are almost identical.

2.4.4 Effect of Optimization Algorithm

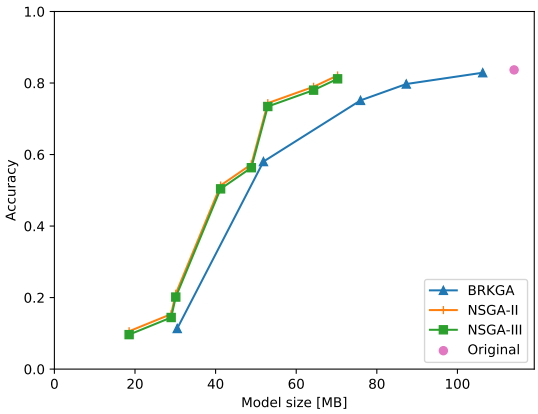


Figure 17: Pareto fronts for different optimization algorithms

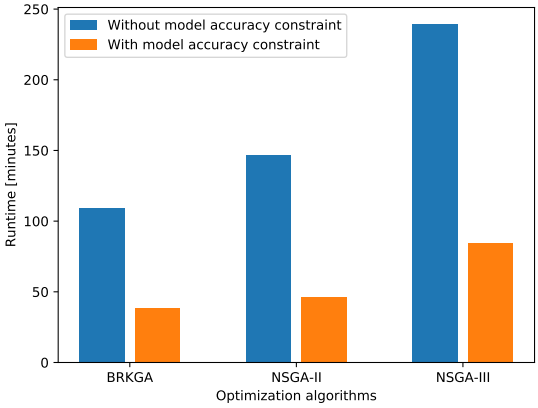


Figure 18: Runtime of different optimization algorithms

To assess the impact of different optimization algorithms, I have evaluated the quality of quantization configurations and runtime with different other optimization algorithms. Here I considered two other multi-objective optimization algorithms: Biased Random Key Genetic Algorithm (BRKGA) [63] and Non-dominated Sorting Genetic Algorithm III (NSGA-III) [64]. Figure 17 compares the Pareto fronts of LeNet5 for BRKGA, NSGA-II and NSGA-III. Evidently, NSGA-II and NSGA-III are able to find better configurations than BRKGA. NSGA-II and NSGA-III output very similar configurations, but NSGA-III takes

longer runtime than NSGA-II (Fig. 18). As well, I observed the same trend when I applied these multi-objective optimization algorithms to VGG16, C3D, LSTM, Bi-LSTM and BERT-xlarge.

2.4.5 Effect of Retraining

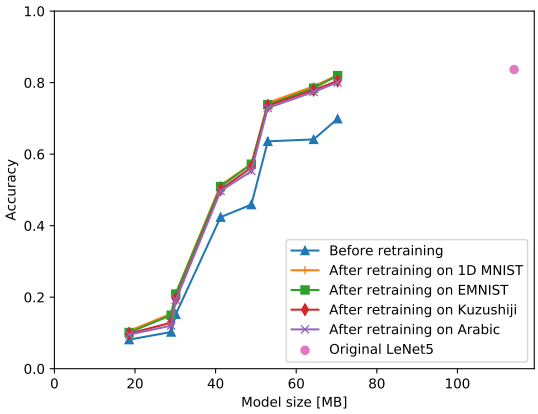
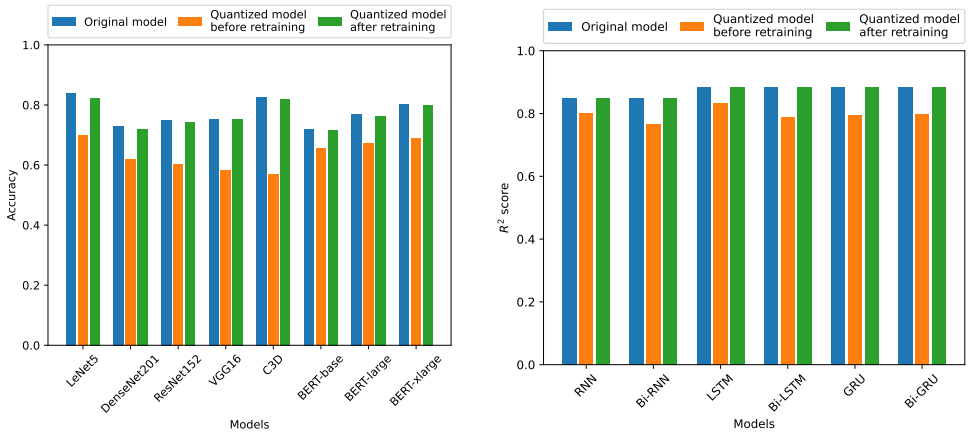


Figure 19: Accuracy of LeNet5 before and after retraining over the different retraining dataset



(a) Accuracy (b) R^2 score

Figure 20: Model performance of the models before and after retraining

I compared the accuracy before and after applying the proposed retraining method to LetNet5 to evaluate if the proposed method improves the accuracy of

the compressed models. Figure 19 presents the Pareto fronts for LeNet5 before and after retraining. As the figure indicates, the retraining process was able to successfully improve the accuracy of LeNet5 with all model sizes. Moreover, I selected the configuration on the Pareto front that achieved the highest accuracy before and after retraining, and plotted the accuracy for all models. The results are shown in Fig. 20a and Fig. 20b. An interesting finding is that smaller models show less improvement of accuracy after retraining. This might suggest that severely quantized small models have lost crucial information that cannot be recovered using retraining.

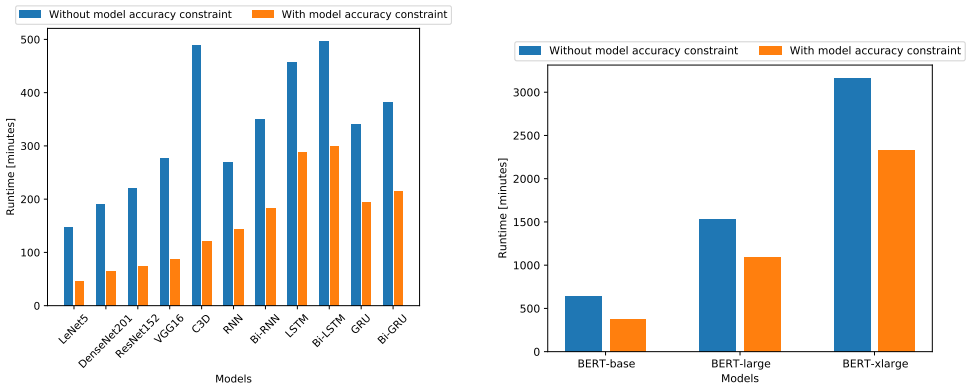
Furthermore, I varied the retraining datasets to observe the impact of different retraining datasets on the model performance after retraining. 1D MNIST, EMNIST [65], Kuzushiji [66] and Arabid [67] datasets are used to retrain LeNet5 after compression. Figure 19 indicates that varying the retraining dataset only slightly affects the accuracy of models. The results indicated that the impact of the retraining dataset to the final accuracy is minimal (less than 3%). Because the proposed method is designed for retraining on unlabeled datasets, the retraining dataset does not have to have the exact same pattern and distribution as the original datasets, the proposed method can retrain the models on different retraining datasets without significant loss of accuracy.

The same trend is observed in other models as well. In summary, the proposed retraining method can effectively improve the accuracy of compressed neural network models.

2.4.6 Runtime

I measured the runtime to find the optimal configurations with and without imposing a constraint on model accuracy. Introducing a constraint is expected to reduce the runtime because it narrows down the objective space to search. Here, I constrain the accuracy of classification and semantic similarity models to their original accuracy minus 10% and the R^2 score of regression models to their original score minus 0.1.

Figure 21a shows the runtime of the proposed method for classification and regression models. Evidently, the runtime becomes faster when imposing constraints. The maximum speedup is achieved with C3D by 75.53%. I found that



(a) Classification and regression (b) Semantic similarity

Figure 21: Runtime of the proposed method with and without accuracy constraint

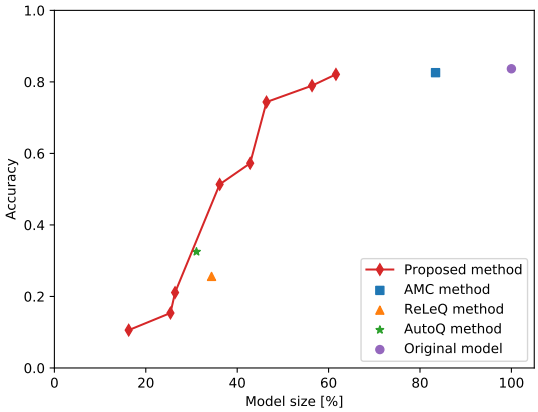
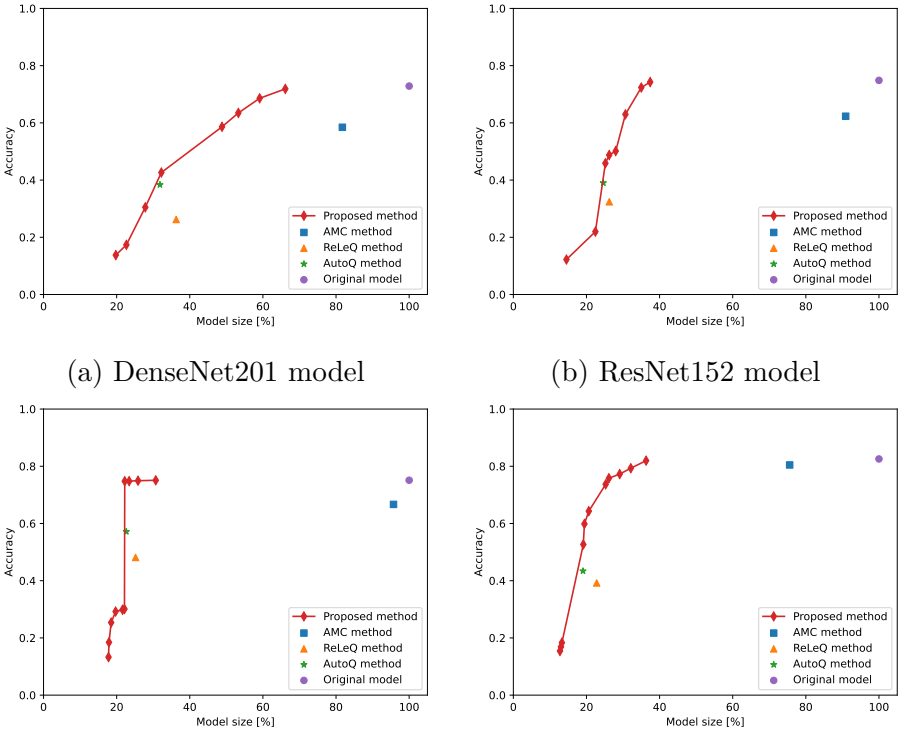


Figure 22: Comparison of proposed and existing methods using LeNet5

classification models benefit from imposing constraints more than the regression models in terms of speedup.

Among the classification models, C3D has the longest runtime because it has the most number of convolutional layers. In contrast, LeNet5 has the shortest runtime because it has the least number of convolutional layers. The number of layers impacts the runtime since the number of configurations that need to be considered is directly proportional to the number of configurations that needs to be optimized by the proposed method.

All regression models have the same number of layers but each model has a



(a) DenseNet201 model

(b) ResNet152 model

(c) VGG16 model

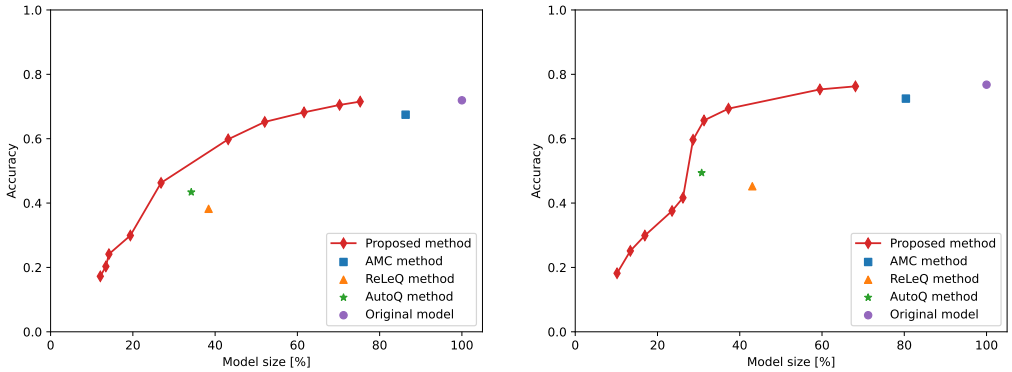
(d) C3D model

Figure 23: Comparison of quantized classification models for different automated quantization methods

different type of recurrent layer. I observed that optimizing bidirectional models takes longer than optimizing unidirectional models. This is because bidirectional models require more time to retrain than the unidirectional models. RNN has the shortest runtime among the regression models since it is simpler than LSTM and GRU because LSTM and GRU are developed based on a simple recurrent neural network layer [68].

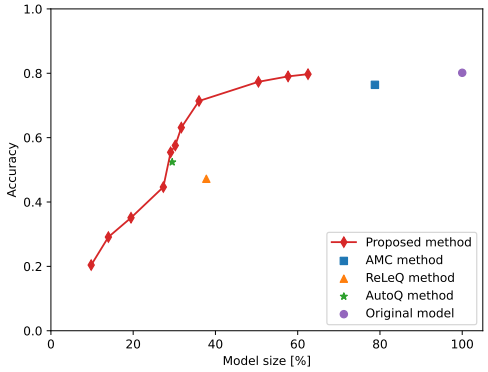
A complex model requires a longer runtime to retrain than a simple model. Since the BERT models are much more complex and larger than the other models, the runtime to retrain these models is also significantly longer than the runtime of the classification and regression models as shown in Fig. 21b.

Therefore, the runtime of the proposed method is mainly affected by the following three factors: (1) the accuracy constraint, (2) the number of layers in



(a) BERT-base model

(b) BERT-large



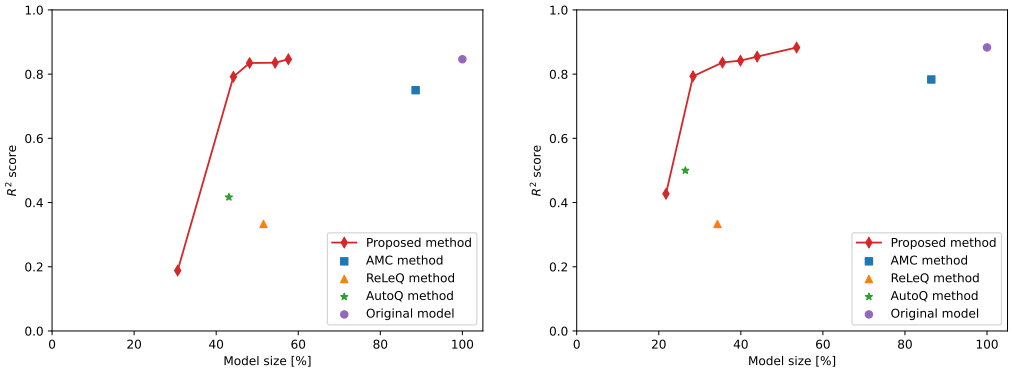
(c) BERT-xlarge model

Figure 24: Comparison of quantized semantic similarity models for different automated quantization methods

the model, and (3) the structure and complexity of the model.

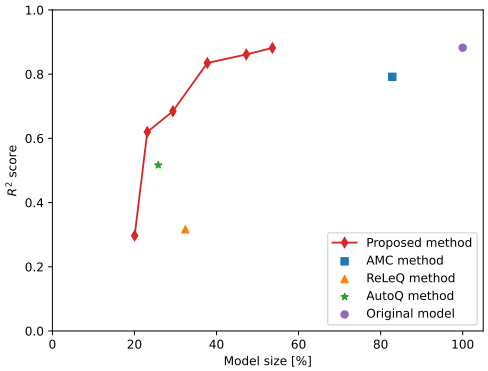
2.4.7 Comparison to Previous Studies

I compared the proposed method to three existing automated quantization methods: AMC, ReLeQ, and AutoQ. The hyperparameters of the existing methods were chosen based on their original set up. The ratio for updating the target model (τ) in AMC was set to 0.01. The threshold for relative accuracy below which the model accuracy loss may not be recoverable (th) in ReLeQ was set to 0.4. The fixed learning rate for the actor and critic networks in AutoQ was set to 10^{-3}. The proposed and existing methods were evaluated on the same



(a) RNN model

(b) LSTM model

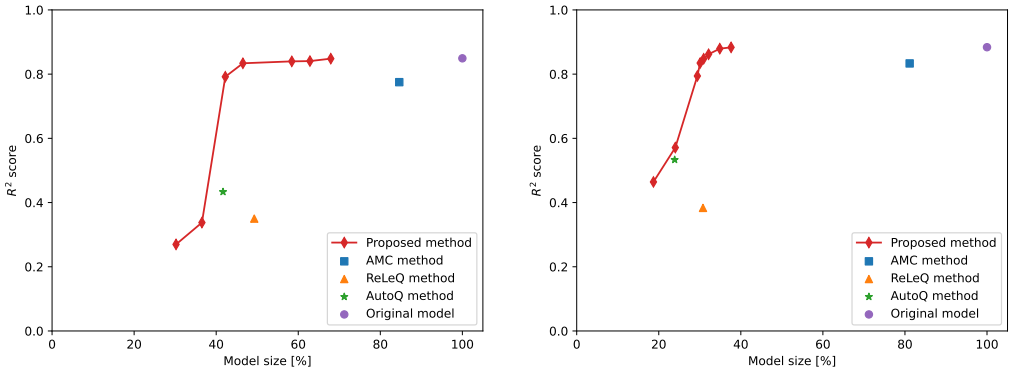


(c) GRU model

Figure 25: Comparison of quantized unidirectional regression models for different automated quantization methods

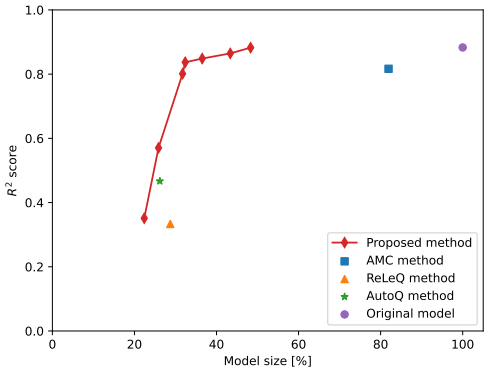
hardware. Also, the configuration of training and retraining the model was the same for the proposed and existing methods including model architecture, training dataset, and retraining dataset. Note that these existing methods provide only a single quantization configuration, but my method provides a set of multiple optimal quantization configurations for multi-objective problems. Moreover, existing methods require the original training dataset to recover the accuracy of quantized models.

Figure 22 plots the results for quantizing LeNet5 using different automated quantization methods. The result clearly reveals that the proposed method was able to find better quantization configurations than the existing methods. The same pattern was observed with other model architectures. The results of other



(a) Bi-RNN model

(b) Bi-LSTM model



(c) Bi-GRU model

Figure 26: Comparison of quantized bidirectional regression models for different automated quantization methods

neural network models using different automated quantization methods are shown in Figs 23, 24, 25, and 26

2.4.8 An Example of a Quantization Configuration

Here, I take LeNet5 as an example and examine one of the quantization configurations suggested by the proposed method. I selected the configuration that achieves the highest accuracy. This configuration reduces the size of LeNet5 from 114.06 MB down to 70.27 MB with just 0.02% loss in accuracy. Table 8 shows the size of each layer before and after compression. LeNet5 is composed of five layers in total: three convolutional layers and two dense layers. The result indicates

Table 8: Size of each layer in the original and compressed LeNet5 models

#	Type	Original size [MB]	Compressed size [MB]	# of centroids
1	Conv.	7.46	-	-
2	Conv.	10.66	-	-
3	Conv.	8.88	-	-
4	Dense	17.41	15.01	128
5	Dense	69.64	30.53	256

that the convolutional layers are kept uncompressed, and the two dense layers are compressed using 128 and 256 centroids, respectively.

Generally, the size of deeper layers is larger than the size of shallower layers. Table 8 indicates that the size of dense layers is larger than the size of convolutional layers in LeNet5. Fully connected layers are amenable to compression without significant loss of accuracy because they have a large number of parameters, and likely to contain redundant parameters.

2.4.9 Discussion

Conventional model compression methods require manual effort to find the optimal configuration. Users need to manually try every possible configuration to find the optimal configuration, which wastes computing resources and runtime. For instance, finding the optimal quantization configuration for LeNet5 using an exhaustive search required evaluating 10^{10} configurations, since there were 10 parameters each with 10 possible levels of quantization (number of centroids). In contrast, the proposed method reduced the number of configurations to be evaluated (10×53) because the optimization converged at the 53rd generations and 10 compressed models were evaluated at each generation.

I also compared the configurations found by the exhaustive search and my proposed method. Figure 27 shows a comparison of the objective space between the exhaustive search and the proposed method to find the optimal configurations for LeNet5. Clearly, the exhaustive search evaluated more configurations than

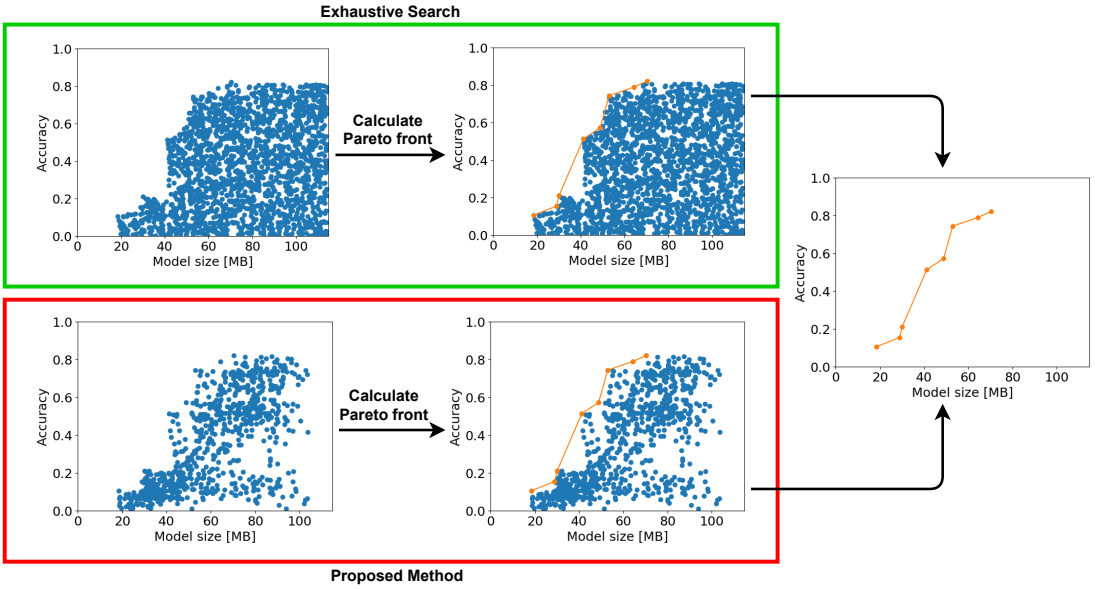


Figure 27: Comparison of objective space between an exhaustive search and the proposed method

the proposed method. Interestingly, the final Pareto fronts completely matched. Therefore, the proposed method can automate to reduce the computing time from the exhaustive search to find the optimal configuration for quantization and for retraining the models. However, there is no guarantee that the proposed method always finds the optimal configurations since it is based on a heuristic search (genetic algorithm).

2.5 Conclusion and Future Work

This chapter proposed a method to automatically find the optimal configurations for quantization and for retraining neural network models without labeled data. The proposed method first compresses the model using vector quantization and then recovers the accuracy using retraining. My retraining method does not require labeled datasets. Furthermore, I use the NSGA-II multi-objective optimization algorithm to automatically find the optimization configuration for quantization that simultaneously minimizes the size and maximizes the accuracy of the model.

The proposed method was evaluated using state-of-the-art neural network models for classification, regression, and semantic similarity tasks. Convergence criteria for the proposed method is the hypervolume, which increases when the number of generations is rises. As a result, the proposed method reduced the size of the models at least 30% while maintaining less than 1% loss of accuracy. Furthermore, retraining without labeled data can successfully improve the accuracy of the compressed model. Lastly, the runtime of the proposed method is related to three main factors: (1) applying the model accuracy constraint, (2) the number of layers which is used to find the optimal centroids, and (3) the structure and complexity of the model which affects the retraining time in the proposed method.

In the future, I plan to assess the generality of my method using a variety of neural networks with diverse structures. I also plan to extend my proposed method beyond neural networks and apply to different machine learning algorithms. Additionally, I will investigate multi-objective optimization methods other than NSGA-II. Various aspects of neural network models, such as memory utilization, inference time, and converging speed, should be examined as optimization objectives.

3. Training Models with Heterogeneous Computing Resources

3.1 Introduction

Federated learning trains a single global model on a centralized server from training data distributed over a large number of edge devices, while it also trains personalized local models on each edge device. The first framework for federated learning was proposed by Google in 2016 [69]. Federated learning has been introduced because of data privacy concerns and data license agreements. For example, the ubiquitous use of cameras in a home environment raises privacy concerns, which is an impediment to install smart home systems [70]. For this reason, edge devices need to have the capability to train neural networks locally [71]. These powerful devices are therefore recently used by many people and produce massive amounts of data including their private information. Thus it is important to use their data while keeping their privacy.

Ensuring data privacy is one of the advantages when applying federated learning because it does not require transferring the local data from edge devices to a centralized server. Furthermore, it reduces the amount of data transfer between the edge devices and the centralized server. The global model is trained by parameters extracted from the local models instead of the local training data itself [72]. There are several types of parameters used in training the global model such as model weights and gradients used to update model weights. In addition to the global model, a personalized local model is built on each edge device by retraining the global model with the local data [73]. The personalized local model is able to integrate both the generic characteristics of datasets on all edge devices and the specific characteristics of the local dataset on each edge device. Users prefer to use the personalized models because the model is tuned for their private information.

However, existing federated learning algorithms such as FedSGD [74] and FedAVG [75] have a critical limitation that assumes the models distributed on the edge devices share the same homogeneous structure. In practical situations, not all edge devices can support the same model due to limitations in available com-

puting resources, storage capacity, physical space, power consumption, network bandwidth and so on. In addition, there are advanced edge devices equipped with accelerators such as GPUs and Google's Edge TPUs[3]. They provide impressive computing performance while occupying less physical space and consuming less power. Each device therefore has different available hardware and limitations. To aggregate information from various edge devices and perform federated learning, I need a method that can handle multiple neural networks with different structures. For this purpose, I look into a method to ensemble heterogeneous models.

In this chapter, I focus on the image classification task. I propose a method based on weighted average to ensemble federated neural networks with heterogeneous model structures. It is reasonable to weight each model differently since the local training dataset may have different amount of data for each output class. Hence I should not use the same weight to average all models. Black box optimization is applied to determine the optimal weight values.

3.2 Background

This section gives a brief overview of existing federated learning methods and combining multiple heterogeneous neural networks.

3.2.1 Federated Stochastic Gradient Descent

Federated Stochastic Gradient Descent (FedSGD) is a federated learning algorithm based on SGD [76]. The global model and local models in FedSGD are trained in the following manner [77]. In each communication round, the centralized server broadcasts the current global model w_t to all clients (t is the current communication round). Each client k then computes the gradient g_k using its local training data and sends the computed gradient to the server. The server averages the gradients received from all clients and generates the new global model w_{t+1} according to Equation 1. Here, η denotes the learning rate, n denotes the the total number of samples, K denotes the total number of clients and n_k denotes the number of training samples on client k.

[3] `https://cloud.google.com/edge-tpu`

$$w_{t+1} = w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k. \tag{1}$$

One of the drawbacks of FedSGD is high communication cost. Since FedSGD needs to frequently communicate between the server and the client [78], it suffers from slow convergence. The most popular approach to tackle this problem is to increase the number of local training epochs and decrease the size of local batches on the client side [79].

3.2.2 Federated Averaging

Federated Averaging (FedAvg) is the current state-of-the-art federated learning algorithm [80]. FedAvg was designed to alleviate the high communication cost and and improve the convergence speed of FedSGD [81]. It increases the number of local training epochs performed on the edge device.

Algorithm 1: Federated Averaging (Server)

```
for t ← 1 to number of communication rounds do
    for k ← 1 to number of selected clients do
        Receive w^k_{t+1} from client k
    end
    w_{t+1} ← Σ_{t=1}^{K} (n_k / n) w^k_{t+1}
    Send w_{t+1} to selected clients
end
```

Algorithm 1 and 2 show the pseudocodes of FedAVG for the server and the client, respectively. On the server side, a subset of participating clients is randomly selected in each round. Next, the server receives the local model w_{t+1}^k from each selected client k. The server then computes the new global model w_{t+1} by averaging the local models received from the clients and broadcasts the new model w_{t+1} to the clients. On the client side, the client receives the latest global model w_t from the server. The client updates the local model w_{t+1}^k using its local training data and sends the updated model w_{t+1}^k to the server. FedSGD can be

Algorithm 2: Federated Averaging (Client)

Receive w_t from server
$w_{t+1}^k \leftarrow w_t$
for $e \leftarrow 1$ **to** *number of local epochs* **do**
    for $b \leftarrow 1$ **to** *number of batches* **do**
        $w_{t+1}^k \leftarrow w_{t+1}^k - \eta g_k$
    end
end
Send w_{t+1}^k to server

thought of as a special case of FedAVG where the local batch size is ∞ and the number of local training epochs is one.

Both FedSGD and FedAVG assume that all edge devices run the same model. Therefore, these existing methods cannot aggregate information from heterogeneous models. This paper aims at overcoming this limitation and proposes a federated learning algorithm that is able to combine models with heterogeneous structures.

While Federated Averaging (FedAvg) is a popular algorithm for federated learning, there are also other algorithms that can be used in this context. There are several ways to perform aggregation in federated learning, including:

- Simple Averaging: The simplest and most common method of aggregation is to take the average of the updated model parameters sent by each client. The server sums the parameters and divides them by the number of clients to obtain the new global model.

- Weighted Averaging: In weighted averaging, each client's update is assigned a weight based on factors such as its training data size or its past performance. The updated model parameters are then combined based on these weights to produce the new global model.

- Federated Averaging with Local Adaption (FedAvgLA): This method is a combination of simple averaging and local adaption. It enables each client to perform additional training on the updated model before sending the

update back to the server. The server then performs simple averaging on the client updates and the locally adapted models to obtain the new global model.

- Secure Aggregation: In situations where data privacy is a primary concern, secure aggregation methods such as homomorphic encryption or secure multi-party computation can be used to ensure that the client updates are not exposed to the server or other clients.

The choice of aggregation method depends on several factors, including the size and complexity of the model, the number of clients, the quality and quantity of data, and the privacy and security requirements of the application.

3.2.3 Combining Heterogeneous Neural Networks

Ensemble learning combines the predictions from multiple models to produce a more accurate prediction than only using one of the models [82]. Ensemble learning has been used in previous works to combine multiple neural networks. Lee *et al.* applied ensemble learning to recognize human actions [83]. They combined multiple LSTM models with different hyperparameters using average ensemble to model both short-term and long-term dependencies. However, they did not consider multiple models with heterogeneous structures.

Deng *et al.* proposed a machine learning model for speech recognition that uses stacking ensemble to combine two models with different structures (an RNN and a CNN) [84]. They assumed traditional centralized learning while this chapter focuses on federated learning.

3.3 Methodology

This section describes the proposed method for ensembling the neural network models with heterogeneous structures.

3.3.1 Overview

The basic idea behind the proposed method is to ensemble the heterogeneous models. Naively applying FedSGD or FedAVG is not possible since there does

not exist any obvious mapping between parameters in different models. Therefore, I employ *weighted average ensemble* as shown in Equation 2. Here, α_{ij} represents the weight for the i-th model and j-th class, x is the input image and y is the final output. I employ weighted average because simple average or majority voting combines all models equally and results in suboptimal performance if the local training datasets distributed across the edge devices are biased.

$$y = \sum_{i=1}^{N} \sum_{j=1}^{C} \alpha_{ij} \cdot m_i(x) \tag{2}$$

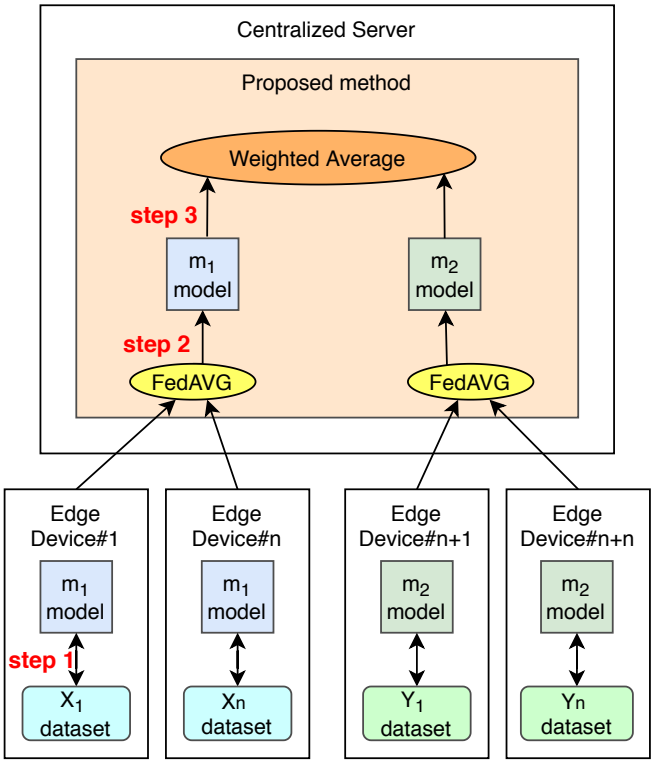


Figure 28: Overview of the proposed method

Figure 28 illustrates the overview of the proposed method. I first deploy different models on each edge device considering its hardware constraints such as processing capability and storage capacity. The edge devices train their models using their local datasets (step 1 in Fig. 28) and send their updated weights to the centralized server. The server aggregates the updated weights for each global

model using FedAVG (step 2). Finally, weighted average ensemble is used to combine the outputs from each model (step 3).

3.3.2 Tuning of Weights

Algorithm 3: Optimization of weights

Input: i-th model m_i, images from the tuning dataset x, labels from the tuning dataset y

Parameters: Number of output classes C, Number of models N

Output: $C \times N$ matrix α where α_{ij} represents the optimized weight for the i-th model and j-th class

1 $\alpha_{ij} \leftarrow \frac{1}{N}$
2 **for** $e \leftarrow 1$ **to** *number of trials* **do**
3 $\quad \alpha \leftarrow$sampling($optimizer$)
4 $\quad$ **for** $d \leftarrow 1$ **to** *number of images* **do**
5 $\quad\quad output \leftarrow \sum_{i=1}^{N} \sum_{j=1}^{C} \alpha_{ij} \cdot m_i(x_d)$
6 $\quad\quad accuracy \leftarrow$validate($y_d, output$)
7 $\quad$ **end**
8 $\quad$ set_parameters($optimizer, \alpha$)
9 $\quad$ set_target($optimizer, accuracy$)
10 $\quad$ optimize($optimizer$)
11 **end**
12 **return** α

The weight α for the weighted average ensemble is tuned by following the procedure as shown in Algorithm 3. I use black box optimization algorithms to tune α. The specific optimization algorithms I consider are shown in the next subsection.

First, α is initialized to a uniform value. In each trial, the black box optimizer suggests a new candidate for α based on the historical data of parameters and achieved accuracy (line 2 in Algorithm 3). The combined output is then computed by multiplying the weight a with the output vectors from each model m_i (line 4). The accuracy of the ensemble model is calculated (line 5) and the obtained

accuracy is fed back to the optimizer (line 7–9). This procedure is repeated for a fixed number of trials.

3.3.3 Optimization Algorithms

1. *Grid Search (GS)*: Grid Search performs an exhaustive search over a given subset of the parameter space. It requires a huge main memory to create the high dimensional parameter space, but it can be easily parallelized since the parameters are usually independent from one another [85].

2. *Random Search (RS)*: Random Search computes all combinations by random selection. It can be generalized to continuous and mixed spaces [86].

3. *Particle Swarm Optimization (PSO)*: Particle Swarm Optimization computes the candidate of parameters by moving the particles around the parameter space based on the position and velocity of the particle. The movement of particle is influenced by its local best target value [87].

4. *Bayesian Search (BS)*: Bayesian Search creates a probabilistic model to search the candidate of parameters. Based on the evaluations of the candidate parameters, it updates the probabilistic model and finds the optimal parameters [88].

5. *Tree Parzen Estimator (TPE)*: Tree Parzen Estimator is similar to bayesian search but it uses a different probabilistic model. It models the best parameter set as a function of the target value, while bayesian search creates a probabilistic model of the objective function [89].

6. *Sequential Model based Algorithm Configuration (SMAC)*: Sequential Model based Algorithm Configuration constructs an explicit regression model to describe the dependence of target value performance on the set of parameters [90].

TPE and SMAC are both sequential model-based optimization (SMBO) algorithms [91]. SMBO builds a probabilistic model for each pair of parameter set and targeted value in each trial, and then the probabilistic model requires to calculate the parameter set for next trials based on the historical pair of parameter

set and targeted value. Thus, the number of historical pairs is increased when the number of trials is increasing. For this reason, SMBO algorithms are very slow but more accurate than GS, RS, PSO and BS, which are not SMBO algorithms.

3.4 Evaluation

This section evaluates the proposed method from four aspects to investigate the characteristics of the ensemble model created with the proposed method. I first evaluate the accuracy achieved with each optimization method along with its runtime. I then investigate if the proposed method can be applied to different combinations of heterogeneous models and datasets.

3.4.1 Experimental Setup

Table 9: Hardware specification

Hardware	Specification
CPU	Intel Xeon E5-2650 v2 (2.20 GHz, 12 cores)
Main Memory	256 GB
GPU	NVIDIA Tesla P100
GPU Memory	16 GB

Table 9 presents the hardware used for the evaluation. Using this server, I simulate the centralized server and all edge devices for the experiments. To evaluate the proposed method, I prepared three experimental setups using two, three and four different models, respectively. Table 10 details of each setup. I evaluated my method with four image classification datasets as shown in Table 11: R-Cellular, CIFAR-10, CIFAR-100 and ImageNet. Since the smallest dataset (CIFAR-10 and CIFAR-100) contains 70,000 images, I divided each dataset into 48,000 images for training, 10,000 images for tuning and 10,000 images for validation. The training dataset is distributed over the edge devices while the tuning and validation datasets are deployed on the centralized server.

I simulated 1,200 edge devices in all setups. In a communication round, 1,000 devices are randomly selected to participate in the federated learning. Each edge

Table 10: Experimental setup

	Setup		
	A	B	C
Total # of devices	1,200	1,200	1,200
# of models	2	3	4
# of devices per model	600	400	300
# of images per model	24,000	16,000	12,000
MobileNet	✓	✓	✓
DenseNet169	✓	✓	✓
ResNet50		✓	✓
VGG16			✓

Table 11: Dataset specification

Name	# of images	# of output classes
R-Cellular	73,000	1,108
CIFAR-10	70,000	10
CIFAR-100	70,000	100
ImageNet	100,000	1,000

device performs 10 local epochs in a communication round and sends its update local model to the server. The server updates the global model and broadcasts the new model to the edge devices. I perform 10 communication rounds in all experiments. Thus, the local batch size is 4 (4 images $\times$ 10 communication rounds). Once the FedAVG step is complete, the optimization step is performed for 50 trials.

I selected four image classification models (MobileNet, DenseNet169, ResNet50 and VGG16) to evaluate the proposed method. Table 12 summarizes the required storage size (MB) to deploy the models and the required number of floating point operations (MFLOPs) in each epoch. Since VGG16 occupies large storage space (553.43 MB) and requires a lot of computation (276.68 MFLOPs), not all edge

Table 12: Model specification

Name	Size [MB]	MFLOPs
MobileNet	17.02	8.52
DenseNet169	57.23	28.77
ResNet50	102.55	51.27
VGG16	553.43	276.68

devices can run VGG16 using their limited resources. On the other hand, if I deploy MobileNet to all edge devices, it would waste the resources of devices that could handle larger models capable of achieving higher accuracy. Through the evaluation, I will confirm that my method efficiently leverages heterogeneous environments and achieves a good accuracy close to that of running VGG16 on all edge devices.

3.4.2 An Example of the Optimized Weights

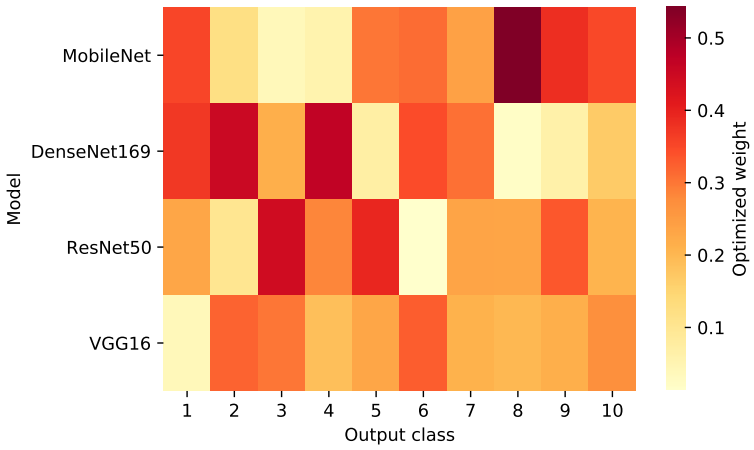


Figure 29: Optimized weights (Setup C, CIFAR-10 dataset and TPE optimization)

Figure 29 visualizes the weights to average the heterogeneous models optimized using the proposed method in the experimental setup C with the CIFAR-10 dataset. For the same output class, models with darker cells are weighted heavier

than the models with lighter cells. Figure 29 clearly reveals that the optimized weight for each output class and each heterogeneous model is different. Taking the 8th output class as an example, MobileNet has the highest weight while DenseNet169 has the lowest weight. This suggests that MobileNet is able to identify the 8th class more accurately than DenseNet169. In this manner, models that achieve higher accuracy on a particular class will have higher weights assigned to them after the weight optimization.

3.4.3 Accuracy of Optimization Methods

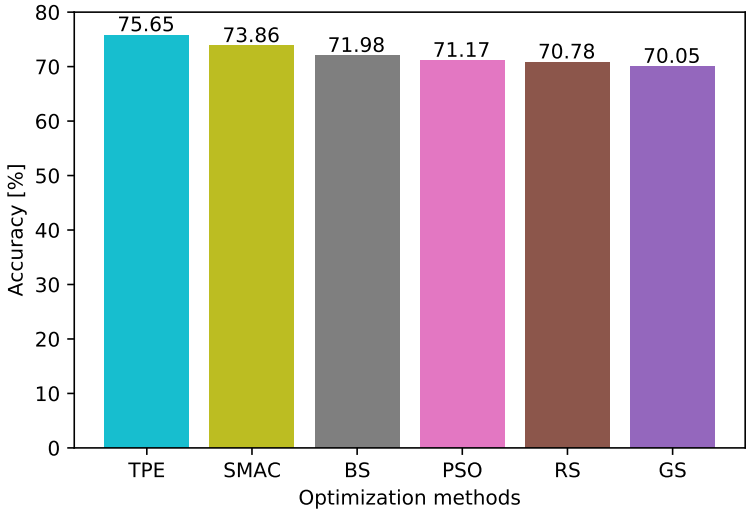


Figure 30: Comparison of accuracy with different optimization methods (Setup A and R-Cellular dataset)

Tuning the weights to ensemble the heterogeneous models is the key in my proposed method. I evaluated the accuracy with respect to the optimization methods for tuning the weights. I used the experimental setup A, which ensembles MobileNet and DenseNet169, and the R-Cellular dataset. Figure 30 shows the achieved accuracy using each of the optimization methods. Figure 31 shows the trends of improvement during the trials. Before applying the proposed method, the accuracy of the combined model was 63.19%. This is effectively averaging the outputs from each model since the weights are uniformly initialized before the optimization. After performing 50 optimization trials, TPE achieved the highest

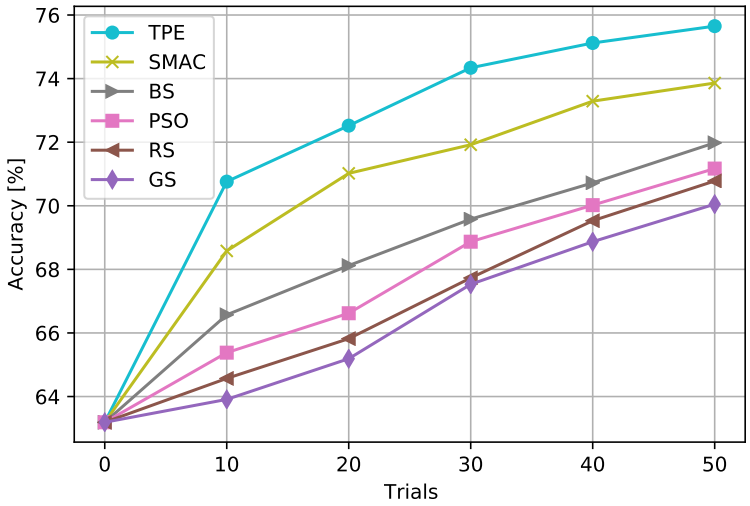


Figure 31: Improvement of accuracy during the trials (Setup A and R-Cellular dataset)

accuracy, 75.65%, while GS produced the lowest accuracy, 70.05%. However, the differences are not that significant. Similar results were also observed with the other experimental setups B and C.

3.4.4 Runtime of Optimization Methods

I compared the runtime required for tuning the weights with different optimization methods. Here I used setup A and the R-Cellular dataset. Figure 32 shows the comparison of runtime. Evidently, optimization methods that achieve higher accuracy require longer runtime. TPE was the slowest taking more than 3 hours to complete the optimization whereas GS was the fastest only taking 17 minutes. I also measured the runtime using experimental setups B and C and observed the same trend. Moreover, I also found out that the runtime increases with the number of models and the number of output classes of each model.

3.4.5 Results for Different Combinations of Models

The purpose of this evaluation is to confirm that the proposed method is still effective if the number of heterogeneous model increases. I measured the accuracy of the proposed method with varying number of heterogeneous models (R-Cellular

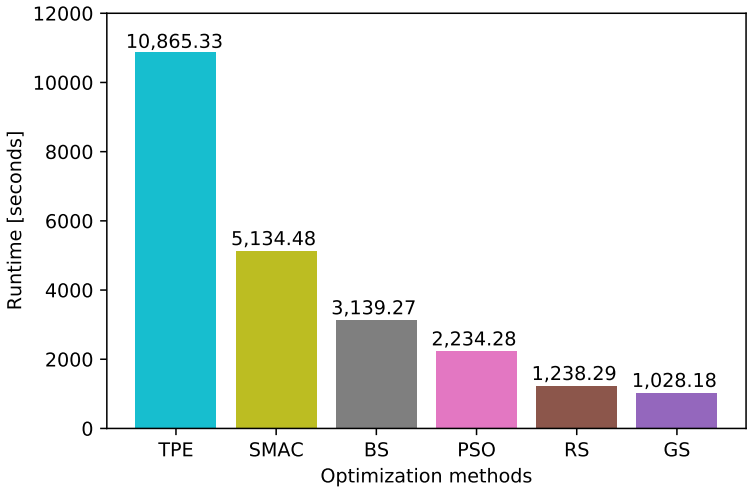


Figure 32: Comparison of runtime with different optimization methods (Setup A and R-Cellular dataset)

Table 13: Comparison of accuracy using different models (R-cellular dataset and TPE optimization)

Model	Centralized	FedAVG
MobileNet	75.43	75.29
DenseNet169	75.94	75.71
ResNet50	76.29	76.14
VGG16	78.57	77.92

dataset and TPE optimization). In addition, I measured the accuracy of each model when trained with a centralized and a distributed dataset using FedAVG as baselines.

Table 13 shows the comparison of accuracy using the different models over centralized and decentralized R-cellular dataset. Table 14 shows the comparison of accuracy using different combinations of models (R-Cellular dataset and TPE optimization). Table 14 indicates that the accuracy of the proposed method in setup A, which combines MobiletNet and DenseNet169, is 75.65%. In FedAVG, MobiletNet and DenseNet169 achieve 75.29% and 75.71%, respectively. This suggests that the proposed method is able to effectively combine its constituent

Table 14: Comparison of accuracy using different combination of models (R-cellular dataset and TPE optimization)

Combination of Models	Proposed
MobileNet + DenseNet169	75.65
MobileNet + DenseNet169 + ResNet50	76.02
MobileNet + DenseNet169 + ResNet50 + VGG16	77.43

heterogeneous models without significant overhead in terms of accuracy. The proposed method can combine the heterogeneous models in setup A, B, and C to 75.65%, 76.02%, and 77.43%, respectively. Furthermore, MobiletNet and DenseNet169 trained with a centralized dataset each reach 75.43% and 75.94% accuracy, which is very close to FedAVG and the proposed method. Thus I conclude that the overhead of federated learning is minimal.

The same trend is observed in setups B and C as well. In summary, the proposed method can effectively combine different combinations of models without incurring significant loss of accuracy.

3.4.6 Results for Different Datasets

Table 15: Comparison of accuracy using different datasets (Setup C and TPE optimization)

Dataset	Centralized	FedAVG	Proposed
R-Cellular	78.57	77.92	77.43
CIFAR-10	89.95	89.67	89.36
CIFAR-100	88.56	88.37	88.29
ImageNet	86.86	86.42	85.88

Lastly, I confirm that the proposed method can be applied to datasets with the different number of output classes. Table 15 shows the accuracy of the proposed method for the four datasets. In addition, the accuracy of VGG16 models trained using centralized and decentralized datasets is shown as baselines. The accuracy

of the ensemble of the four models (77.43%) is very close to a VGG16 model training using FedAVG (77.92%). The difference of accuracy between VGG16 and the proposed method is consistently less than 1% for all datasets. The accuracy of the proposed method is even comparable to a VGG16 model trained using a centralized dataset.

3.4.7 Discussion

The proposed method was evaluated by varying the number of output classes in the datasets and the number of models with heterogeneous structures. I found that the runtime of the proposed methods increases with the number of classes in the dataset and the number of heterogeneous models. This is because the proposed method needs to iterate over every output class of every heterogeneous model.

I found that TPE achieves the highest accuracy among the six optimization methods I compared. On the other hand, TPE takes the longest runtime. This is likely because TPE defines the parameter space as a tree structure, thus it requires longer time to set up the parameter space and to traverse the tree structure to select the parameter to optimize. However, TPE is the best parameter optimization method in terms of accuracy because the tree traversal is able to to select the optimized parameter and lookup the optimized parameter in the next trial by focusing on the same sibling.

3.5 Conclusion and Future Work

In this chapter, I proposed a novel federated learning algorithm for neural network models with heterogeneous structures. The proposed method utilizes weighted average ensemble to combine the outputs from different models. The proposed model aggregator allows for combining models with heterogeneous structures, which enables training on different input formats. However, to perform ensembling, the output format of the training dataset must be the same. If clients train their models using different output formats, it may lead to inconsistencies in the proposed method. The proposed model aggregator component is enabled gathering data from various data sources since it allows for combing the model

with the diverse structures from heterogeneous computing resources. Thus, the proposed model aggregator component can extend data diversity to training the machine learning models with a federated learning approach.

Moreover, Black box optimization is used to tune the weights for the output classes of each model. I compared six black box optimization algorithms (grid search, random search, particle swarm optimization, bayesian search, tree Parzen estimator and sequential model-based algorithm configuration) and found that TPE was able to achieve the highest accuracy. However, TPE took the longest runtime among the six methods.

As a future work, I will enhance the proposed method to regression and localization tasks. In addition, other parameter optimization techniques and federated learning algorithms will be investigated. Due to the current implementation of the proposed model aggregator component, the ensembled model does not return to the clients but the aggregated model is returned to the clients. The size of the ensembled model is increased so that users need additional resources to execute the ensembled model. In the future, I will propose a method for building the ensembled model to be able to execute on the client ' s hardware resources.

4. Training Models with Heterogeneous Network Resources

4.1 Introduction

Currently, there has been a rise in the use of edge devices since edge devices have more computing power than ever [92]. Massive complex structured and unstructured data are being generated and accumulated over a large number of edge devices [93]. The collected data on mobile devices is advantageous for deep learning, which requires a significant amount of data [94]. However, transferring raw data directly from clients to a server is not a preferred procedure as it lacks data privacy and consumes a lot of network resources [95]. Thus, federated learning has been introduced to address these issues. Federated learning is a machine learning method to train a single global model on a centralized server by indirectly using training data distributed across a large number of edge devices. In federated learning, the clients transfer only the trained local models instead of the raw data, and it mitigates privacy and network communication issues [96]. It also allows each edge device to train a client-specific local model, while training the global model [97].

Nevertheless, there is still room for improvement in reducing the communication cost of federated learning, since just transferring the trained models still consumes a significant amount of network resources [98]. Federated learning is executed over a large number of edge devices that are often connected to wireless networks such as cellular networks. Since wireless networks provide lower bandwidth compared to wired networks, communicating the models between the server and clients takes longer and thus limits the frequency of model updates [99]. Thus, by improving the communication efficiency, the global model on the server can be continuously and efficiently updated, and the performance of federated learning can be improved [100].

There are two popular approaches to reduce the required communication cost for federated learning: (1) reducing the number of communication rounds [101], and (2) reducing the amount of communication in each round [102]. However, if the number of communication rounds is reduced, the global model may miss local

training information that could have been obtained in the omitted rounds, and thus the model cannot be continuously updated [103]. Reducing the communication cost in each round can also be divided into two approaches: (1) transferring whole models from selected clients [104] and (2) transferring compressed models from all clients [6]. In particular, transferring compressed models is a promising approach because it can balance the trade-off between communication cost and model accuracy by adjusting the compression ratio [105].

In this chapter, I propose a method to transfer sparse models instead of dense models for reducing the communication cost on both uplink and downlink in federated learning. The proposed method constructs a sparse model by selecting only parameters that have been updated significantly. I compute the absolute difference between the parameters of the local model before and after training, and exchange only the upper quantile of the updated parameters between the server and the clients. This parameter-wise selection approach increases the opportunity to reduce the communication cost since it omits the unnecessary parameters and keeps the necessary parameters for transfer. It is reasonable not to transfer the parameters that do not have significant updates in the local model, since they may not have much impact on the global model update. Additionally, the proposed method allows adjusting the trade-off between the model accuracy and the communication cost with a hyperparameter. This hyperparameter controls the level of sparsification, *i.e.* what fraction of the model are exchanged between the server and clients.

4.2 Background

This section gives a brief overview of existing techniques to reduce the communication cost in federated learning.

Edge computing is a backbone of the federated learning then the technical challenges of edge computing are also the challenges of federated learning [106]. Computing on edge devices should consider the limitation of client's resource constraints and unstable network communication [107]. I focus on the challenge of running federated learning on unreliable and asymmetric connections due to wireless network connection. Federated learning applications are often executed over slow and unstable internet connections as WiFi [108]. Federated learning

Algorithm 4: Federated averaging (ServerExecutes)

Parameter: R is the number of communication rounds, N is the number of clients, C is the fraction of selected clients, w_r^n is the local model for client n at round r, w_r is the global model at round r

Initialize w_0
for $r = 1, 2, \ldots, R$ **do**
  $S_r \leftarrow$ A random set of $C \times N$ clients
  for $n \in S_r$ **do**
    $w_{r+1}^n \leftarrow$ **ClientsUpdate**(w_r)
  end for
  $w_{r+1} \leftarrow \frac{1}{N} \sum_{i=1}^{N} w_{r+1}^i$
end for

algorithm is an algorithm to describes transferring and updating the local and global models [109].

4.2.1 Federated Learning Algorithms

Federated Stochastic Gradient Descent (FedSGD) is a popular conventional federated learning algorithm based on SGD [76]. The state-of-the-art of federated learning algorithm is Federated Averaging (*FedAVG*) [110]. FedAVG was developed to reduce the communication costs of FedSGD and improve the convergence speed [81]. The number of local training epochs performed on the client is increased compared to FedSGD.

Algorithm 4 and 5 show the pseudocodes of FedAVG for the server and the client, respectively. On the server side, a subset of participating clients is randomly selected in each round. Next, the server receives the local model w_{r+1}^n from each selected client n. The server then computes the new global model w_{r+1} by averaging the local models received from the clients. On the client side, the client receives the latest global model w_r from the server. The client updates the local model w_{r+1}^n using its local training data. Here, η and g denote the learning rate and the gradient, respectively. Each client sends the updated model w_{r+1}^n to the server. FedSGD can be considered as a special case of FedAVG where the

Algorithm 5: Federated averaging (ClientsUpdate)

Parameter: E is the number of epochs, B is the number of batch sizes, w_r^n is the local model for client n at round r

Receive w_r from the server
Initialize $w_{r+1}^n \leftarrow w_r$
for $e = 1, 2, \ldots, E$ **do**
    for $b = 1, 2, \ldots, B$ **do**
        $w_{r+1}^n \leftarrow w_{r+1}^n - \eta g_n$
    end for
end for
Transfer w_{r+1}^n **to server**

local batch size is ∞ and the number of local training epochs is one. In addition, FedAVG allows assigning the fraction of selected clients to reduce the number of participated clients in each communication round.

4.2.2 Reducing Communication Costs in Federated Learning

I categorize the existing methods for reducing communication costs in federated learning into two approaches: (1) transferring uncompressed models from selected clients and (2) transferring compressed models from all clients.

The most common approach for transferring the whole model from selected clients is constructing the criteria to decide which clients should be selected. Wu et al. proposed a method to measure the similarity between the global model and the local model on each client using the gradient distribution [111]. They scored each client and adjust its participation probability of each client using the similarity score between the local and global models. Park et al. proposed *FedPSO* [112] to reduce the number of selected clients in each round using particle swarm optimization. They used particle swarm optimization to select the clients based on their loss values of local training. However, the loss value is not a good performance indicator for representing the performance of a local model since the local training datasets are often not identically distributed across the clients.

Yao et al. proposed *FedMMD* [113] to select the clients that participate in

each communication round using maximum mean discrepancy, a distance measure in the probability space. They computed the maximum mean discrepancy of a local model before and after the local training, and transfer the local model to the server only if the maximum mean discrepancy reaches a threshold.

Compared to the method of deciding whether to select or not on a client-by-client basis, the model compression methods can select data to be transferred at a finer granularity. We, therefore, focus more on the model compression methods for federated learning in this paper. Konnecy et al. worked on compression of transferred models using techniques such as *Low Rank Approximation*, *Random Masking*, and *Quantization* [105]. However, they compressed uplink communication only since the uplink is typically slower than the downlink. In contrast, my method considers reducing communication cost in each round by transferring the compressed model from all clients, and also reducing the communication cost of both uplink and downlink.

Sparsification and compression are both techniques used to reduce the size of machine learning models and minimize communication costs in federated learning. In the context of federated learning, sparsification is preferred over compression at the beginning step because the initialized global model needs to be distributed to every client without sparsification. After every client has an initialized local model, sparsification can be applied to reduce the communication cost between a server and clients. Since sparsification focuses on eliminating parameters, it can significantly reduce communication costs compared to compression techniques, which only reduce redundant parameters.

Aji et al. introduced sparsification of models for distributed learning [114]. Their proposed method exchanges only the top updated parameters of the global and local models. The server then transfers the same sparse global model to all clients. However, I propose a method to exchange different sparse models for each client in order to improve the performance of federated learning in terms of both communication efficiency and accuracy. The main difference between distributed learning and federated learning is that distributed learning proposes to achieve high accuracy over a single dataset, while federated learning proposes to achieve high accuracy over multiple datasets across the clients. Therefore, my proposed method transfers different sparse models in order to optimize the models to the

local dataset on each client.

4.3 Methodology

The basic idea behind the proposed method is to sparsify the models exchanged between the server and clients. Parameters in neural networks are usually constructed as a graph data structure where each node does not update its weight as others [115]. The most updated parameters might have the most impact on model performance since it affects the cumulative changing of the graph more than the less updated parameters. Thus, I compute the absolute difference of model parameters before and after local training, and construct a sparse model that contains only the upper quantile of updated parameters. The server constructs a sparse global model for each client with the same sparsification pattern as the local model received from the client, and sends it to the client.

4.3.1 Server Executes

Algorithm 6 shows the pseudocode of the proposed method on the server. At the initialization step, the server distributes the global model to all clients. The server then receives the sparse local model from each client (line 4). The sparse local models are aggregated to a dense global model using averaging (line 6). Next, a sparse global model is created by replacing the parameters of the sparse local model received from the client with the updated parameters of the global model (line 10). The same sparsification pattern is maintained by leaving the dropped parameters (p is NULL) in the local model as NULL (line 12). The sparse local model is compressed using gzip[4].

4.3.2 Clients Update

Algorithm 7 shows the pseudocode of the proposed method on the clients. Each client receives the sparse global model from the server (line 1). Next, a dense local model is constructed by replacing the NULLs in the received sparse global model with the dense local model from the previous round (line 2–7). The dense local model is then trained using the local dataset (line 8–12). I then

[4]https://www.gnu.org/software/gzip/

Algorithm 6: Proposed method (ServerExecutes)

1 **Parameter**: R is the number of communication rounds, N is the number of clients, w_r^n is the local model for client n at round r, w_r is the global model at round r

1: Initialize w_0
2: **for** $r = 1, 2, \ldots, R$ **do**
3: **for** $n = 1, 2, \ldots, N$ **do**
4: $w_{r+1}^n \leftarrow$ **ClientsUpdate**(w_r^n)
5: **end for**
6: $w_{r+1} \leftarrow \frac{1}{N} \sum_{i=1}^{N} w_{r+1}^i$
7: **for** $n = 1, 2, \ldots, N$ **do**
8: **for** $p \in$ parameters of w_{r+1}^n **do**
9: **if** p is not NULL **then**
10: $w_{r+1}^n[p] \leftarrow w_{r+1}[p]$
11: **else**
12: $w_{r+1}^n[p] \leftarrow$ NULL
13: **end if**
14: **end for**
15: **end for**
16: **end for**

compute the threshold for sparsification. I compute the absolute differences of parameters before and after local training, and use their Q-quantile value as the threshold (line 13). The parameter Q is supplied by the user to adjust the communication cost and model accuracy. For instance, if Q is set to 0.9, the top 10% of the parameters are selected for transfer. Afterwards, a sparse local model is constructed by replacing the parameters less than the threshold with NULLs (line 14–18). The sparse local model is then compressed using gzip. At last, the compressed sparse local model is sent to the server (line 19).

In sparse distributed learning, the same global model is broadcasted to all clients. In constrast, my proposed method for federated learning sends the global model to each client with different sparsification patterns. In my method, the sparsification pattern of the global model depends on the sparsification pattern

Algorithm 7: Proposed method (ClientsUpdate)

Parameter: E is the number of epochs, B is the number of batch sizes, Q is the quantile, w_r^n is the local model for client n at round r

Receive w_r^n from the server
for $p \in$ parameters of w_r^n **do**
  if p is NULL **then**
    $w_r^n[p] \leftarrow w_{r-1}^n[p]$
  end if
end for
Initialize $w_{r+1}^n \leftarrow w_r^n$
for $e = 1, 2, \ldots, E$ **do**
  for $b = 1, 2, \ldots, B$ **do**
    $w_{r+1}^n \leftarrow w_{r+1}^n - \eta g_n$
  end for
end for
threshold $\leftarrow$ **Quantile**$(|w_r^n - w_{r+1}^n|, Q)$
for $p \in$ parameters of w_{r+1}^n **do**
  if $|w_r^n[p] - w_{r+1}^n[p]| <$ threshold **then**
    $w_{r+1}^n[p] \leftarrow$ NULL
  end if
end for
Send w_{r+1}^n to the server

of the local model received in the previous communication round. It means that only the parameters sent to the server due to large updates confirmed in the local training will be updated back with the parameters of the aggregated global model. This prevents unnecessary updates of the local model and also helps the local model maintain high accuracy on its local dataset.

4.4 Evaluation

This section evaluates my proposed method from four aspects. First, I compare the proposed method to the sparse communication method for distributed

Table 16: Experimental Setup

Configuration	Value
# of communication rounds (R)	10
# of clients (N)	10
# of local epochs (E)	5
# of local batch sizes (B)	8

Table 17: Model specification

Name	Size [MB]	# of Parameters
VGG16	553.43	138,357,544
ResNet152	243.21	60,419,944
DenseNet201	82.92	20,242,984
MobileNet	17.02	4,253,864

learning. Second, I compare the proposed method to existing communication reduction methods for federated learning. Third, I evaluate the proposed method using four state-of-the-art neural network models for image classification task to assess the impact of the model size on the performance of the proposed method. Lastly, I investigate if the proposed method can be applied to different datasets.

4.4.1 Experimental Environment

I measured the communication cost and accuracy of the global model to evaluate the practical applicability of the proposed method. The communication cost on both uplink and downlink communication was measured by estimating the number of bytes transferred over the network connection. All sparse global and local models were stored for each communication round. Then, for each communication round, I calculated the size of each sparse local model and the size of sparse global model multiplied by the number of participating clients to estimate the total number of transferred bytes. The experimental setup for federated learning is shown in Table 16. The experimental environment was set up using Docker,

and the server and clients were executed in separate containers.

I conducted my experiments using four neural network models and four datasets. Table 17 shows the specifications of the models used to evaluate the proposed method, VGG16, ResNet152, DenseNet201, and MobileNet. These four models represent state-of-the-art image classification models of different scales. Although large-scale models can achieve higher accuracy than small-scale models, not all edge devices can deploy large scale models due to resource constraints. Thus, I evaluated the proposed method using models with different scales.

The experiments were conducted using CIFAR10, CIFAR100, MNIST, and FMNIST datasets. These four datasets are the most popular datasets used to evaluate image classification tasks. CIFAR10 and CIFAR100 consist of 60,000 images (50,000 training samples and 10,000 testing samples) each of which is a 32×32 pixel 3-channel image. MNIST and FMNIST consist of 70,000 images (60,000 training samples and 10,000 testing samples) each of which is a 28×28 pixel single-channel image. The training samples are distributed equally to each client, and the testing samples are stored on the server to evaluate the accuracy of the global model.

4.4.2 Comparison to Distributed Learning

I compared the performance of the proposed sparse communication method for federated learning and that of the existing method for distributed learning [114]. Here, distributed learning refers to the setup where the server sends the same sparsified model to all clients, whereas in the proposed method the server sends a different sparsified model to each client. While distributed learning focuses on training a global model to achieve high accuracy over a single dataset, federated learning is expected to maintain the accuracy of both global and local models over the global and local datasets. Thus, I measured the accuracy of global model and the average accuracy of local models between my federated learning and the existing distributed learning. The global model was evaluated using the testing samples on the server and the local model was evaluated using the training samples on its own local device.

Figure 33 presents the accuracy of global model and average accuracy of local models between federated learning and distributed learning in the case of VGG16

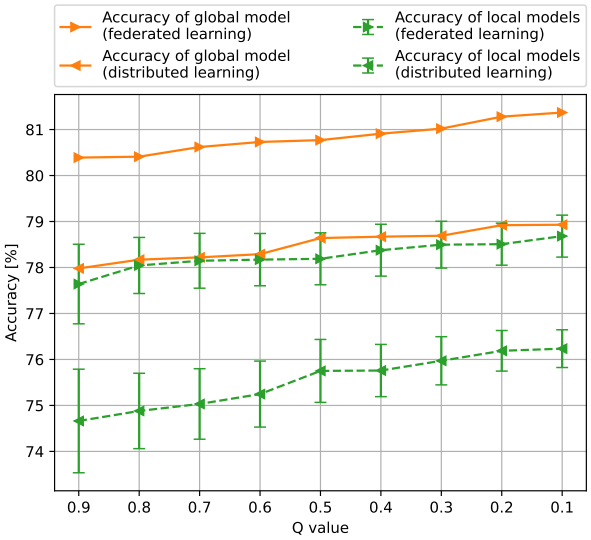
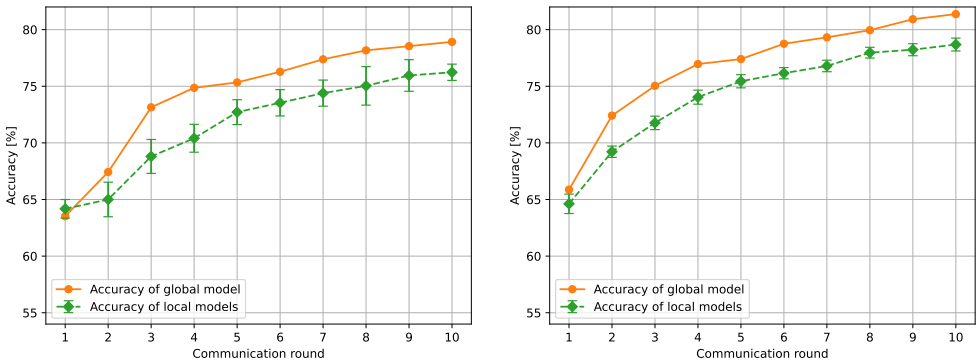


Figure 33: Accuracy of global and average accuracy of local models between federated and distributed learning

model on CIFAR10 dataset while varying the value of the hyperparameter Q, which adjusts the communication cost. I found out that both the accuracy of the global model the average accuracy of local models of my proposed method are higher than that of the existing distributed learning. Moreover, I found out that the variance of local model accuracy of the distributed learning is larger than that of my method as shown in Fig. 33.

Figure 34 shows the accuracy of global and local models at each communication round for the existing distributed learning and my federated learning. In the distributed learning, I observed a large variance in the accuracy of the local model during the training process as shown in Fig. 34a. The global model with the same sparsification pattern used in distributed learning reduces the accuracy of the local model because the local model is initialized with the generalized global model for each communication round. On the other hand, the accuracy of local models of my federated learning is more stable than that of distributed learning as shown in Fig. 34b. In the federated learning, the accuracy of global model is improved continuously unlike in the distributed learning.

Although distributed learning, which aims to achieve high accuracy with only a single global dataset, is expected to achieve higher accuracy in the global model,



(a) Distributed learning (b) Federated learning

Figure 34: Comparison of sparse communication methods for federated and distributed learning ($Q = 0.1$)

the results show that my federated learning model achieved higher accuracy in the global model as well. This may be due to the fact that distributed learning, which transfers the same global model to all clients, is not able to build highly accurate local models, which also leads to a decrease in the accuracy of the global model.

4.4.3 Comparison to the Existing Methods

I compared the performance of the proposed method and existing learning methods for communication reduction: FedAVG, FedPSO, FedMMD, Low rank approximation, Random masking, and Quantization. Figure 35 shows the relative communication cost and global model accuracy for VGG16 model on CIFAR10 dataset. Each method can be adjusted for accuracy and communication reduction by hyperparameters. Table 18 shows the hyperparameters of each method. The original communication cost indicates the total communication cost when FedAVG is applied with C set to 1.0.

FedPSO, FedMMD, Low rank, Random mask, and Quantization do not reduce communication cost by more than 50% since they reduce the uplink communication only. FedAVG and the proposed method are the only two methods that can reduce the communication cost by more than 50%. However, the accuracy of the global model in FedAVG decreases sharply when reducing the fraction of

Table 18: Learning methods and hyperparameters

Name	Hyper-parameter	Description
FedAVG	C	The fraction of clients selected in each communication round. All clients will participate in the communication round if C is set to one.
FedPSO	α	The inertia weight of swarm optimization for each client during optimization. Varying α value does not reduce the communication cost. I set α to 0.1 which produce the highest accuracy.
FedMMD	λ	The coefficient of MMD loss between the global and local models. The clients have less MMD loss compare than the previous round will be selected.
Low rank approximation	K	The rank of the low-rank matrix to be converted.
Random mask	M	The size of random mask to generate a random sparsification pattern.
Quantization	B	The quantized bit used for bit-quantization.

the selected clients. On the other hand, the accuracy of the global model in the proposed method decreases slowly when increasing Q.

The results show that the proposed method outperforms FedAVG in terms of the required communication cost and the accuracy of the global model. FedAVG decreases the number of selected clients to reduce communication cost, and thus some local datasets are not used in the training. On the other hand, the proposed method does not eliminate the number of clients, but instead replaces the transfer of dense local models with sparse local models, and removes parameters that are less updated in the model.

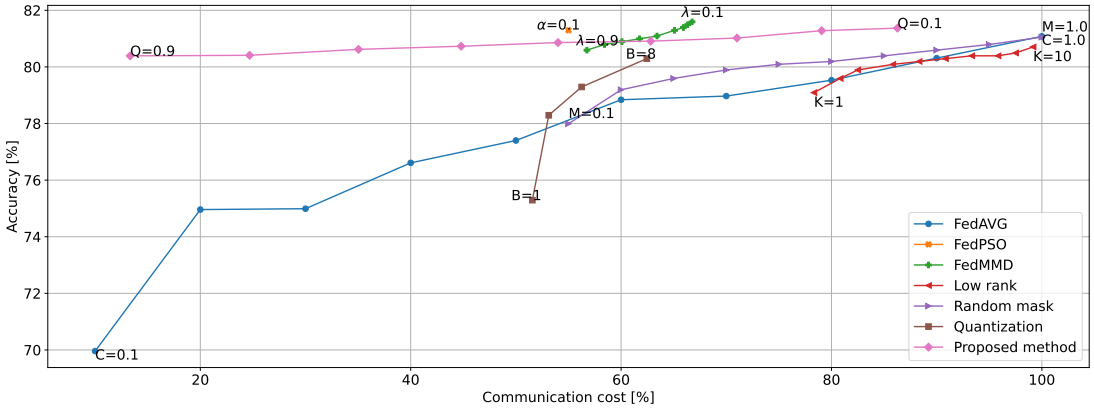


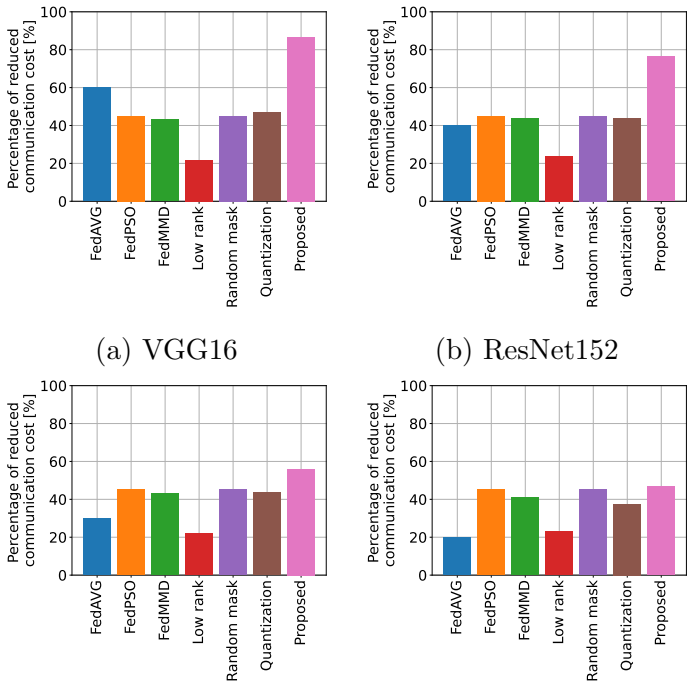
Figure 35: Accuracy of global model and total communication cost (VGG16 model and CIFAR10 dataset)

4.4.4 Results for Different Models

I investigated the impact of model size on the reduction of the required communication cost and global model accuracy using the proposed method and the existing methods. The experiments were conducted for the model with different size on the CIFAR10 dataset.

Figure 36 presents the optimal results of the communication cost reduction ratio of the proposed method and the existing methods for four image classification models (VGG16, ResNet152, DenseNet201, and MobileNet) when the accuracy of the global model satisfies an acceptable accuracy. Here, the acceptable accuracy is defined as 5% lower than the original accuracy. The results indicate that the reduction of the communication cost from FedPSO, FedMMD, Low rank approximation, Random mask, and Quantization are almost identical for all model architectures. Low rank approximation achieves the least reduction when compared to other methods.

FedPSO and FedMMD reduce the communication costs by building criteria to reduce the number of clients selected to only those that fulfil the criteria. The number of selected clients based on the criteria will be a small number as one to three clients in each communication round. This number of selected clients does not vary much in FedPSO and FedMMD. Therefore, the communication cost of those methods does not vary much either since the variation of the communica-



(a) VGG16 (b) ResNet152

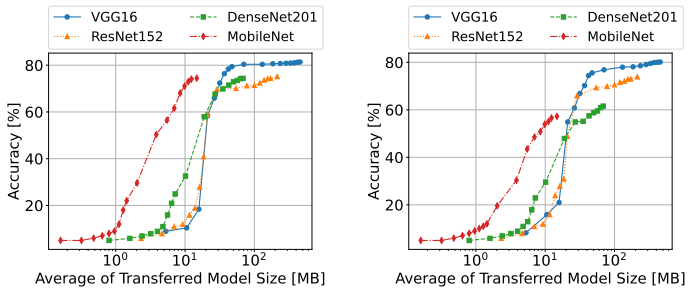
(c) DenseNet201 (d) MobileNet

Figure 36: Percentage of reduced communication cost for different models

tion cost depends on the number of selected clients. On the other hand, Low rank, Random Mask, and Quantization are compression techniques that do not consider the model architecture, and the compression ratio depends on the number of redundant parameters, not the model architecture. Thus, the reduction in communication costs through these methods is similar for all models. However, the communication cost reduction by FedAVG and the proposed method depends on the model architecture. The communication cost reduction by the proposed method outperforms the others in VGG16 and ResNet152, but not so much in DenseNet201 and MobileNet.

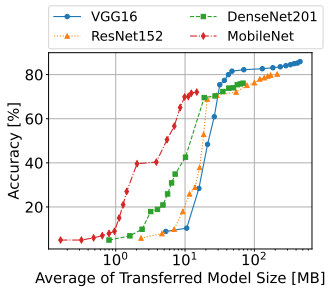
4.4.5 Results for Different Datasets

In this evaluation, I measured the average of transferred model size against the accuracy of the global model by varying the dataset since I aim to investigate the performance of the proposed method to reduce communication cost on dif-

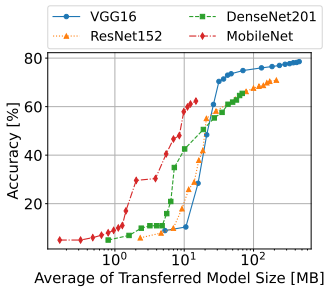


(a) CIFAR10

(b) CIFAR100



(c) MNIST



(d) FMNIST

Figure 37: Average transferred model size for each model architecture with the different datasets

ferent model architectures and datasets. The average of transferred model size represents how much the proposed method is able to reduce the size of each architecture model for transferring between a server and the clients.

Figure 37 shows the global model accuracy and average transferred model size for different model architectures and datasets. Each line connects 18 markers representing different Q values. In this evaluation, I varied Q from 0.1 to 0.9 at an interval of 0.1, and 0.91 to 0.99 at an interval of 0.01. In the case of CIFAR10, the proposed method reduced the average transferred model size of VGG16 from 553 MB to 32 MB (94.21% of reduction), while the global model accuracy was 76.39%. This is slightly (1.93%) higher than the global model accuracy of MobileNet when the average transferred model size is 17 MB as shown in Fig. 37a. If the proposed method reduces the average transferred model size of VGG16 with the same reduction size in the case of CIFAR100 dataset then the global model accuracy of VGG16 is significantly higher than the global model accuracy of MobileNet

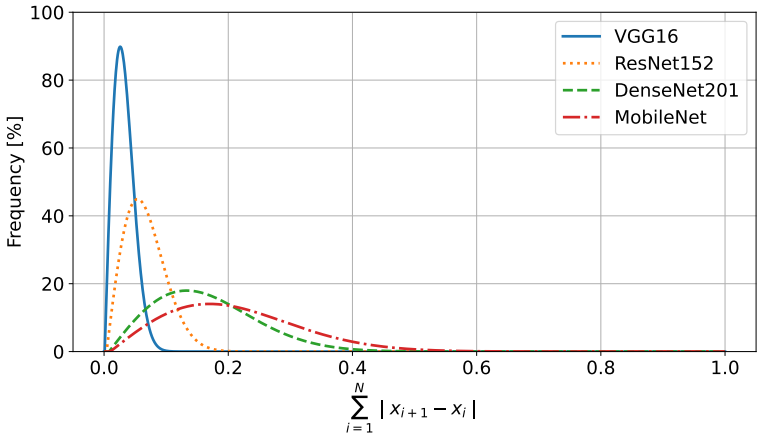


Figure 38: Distribution of parameter updates for each model

by 9.64% as shown in Fig. 37b. In addition, I observed that the global model accuracy of VGG16 is slightly higher than the global model accuracy of MobileNet by 3.32% in the case of MNIST dataset but the global model accuracy of VGG16 is significantly higher than the global model accuracy of MobileNet by 9.10% as shown in Fig. 37c and Fig. 37d.

Application requirements and resource limitations need to be considered when selecting the model to deploy on the clients and server. The selected model must meet the requirements in terms of computing, storage and network resources of every edge device, and achieve the model accuracy required by the application. Typically, a larger model is able to achieve accuracy higher than a smaller model. The proposed method can reduce the communication cost of a large model until consume the network resources slightly larger than a small model while maintaining significantly higher accuracy.

4.4.6 Distribution of Parameter Updates

The proposed method reduces the communication cost of a large model more than a small model with most models. Figure 38 plots the frequency of updated values per communication round and per client for each model architecture. I found out that the frequency of small updated parameters in a larger model is higher than in a smaller model. Updating parameters in neural network models represents how much the model changes, which means small models require more

updating than large models [116]. Therefore, the proposed method can reduce the communication cost of large models more than small models since large models update the parameters with the lower threshold of parameters than small models.

Generally, a larger model achieves higher accuracy than a smaller model [117]. However, the proposed method reduces the communication cost of a large model to be slightly larger than of a small model while a smaller model has slightly lower accuracy on CIFAR10 and MNIST datasets. Furthermore, I found out that the proposed method is able to reduce the communication cost of a large model to be slightly larger than of a small model while a smaller model has significantly lower accuracy on CIFAR100 and FMNIST datasets. Hence, deploying the model on the edge devices for federated learning depends on the use case scenario or resource constraints.

4.5 Conclusion and Future Work

In this chapter, I proposed a novel method to reduce the communication cost for federated learning on both uplink and downlink communication. The proposed method utilizes exchanging the most updated parameters of neural network models. I used diverse models and datasets to evaluate the proposed method in terms of model accuracy and communication cost. The proposed method achieved a reduction in the communication costs approximately 90% compared to the traditional method for VGG16. Moreover, I found out that the proposed method reduces the communication cost of larger models more than smaller models since the threshold of updated parameters in a large model is greater than in a small model. In some cases, a small model achieve slightly lower accuracy than a larger model with a slight difference in communication cost.

A future work is to investigate other neural network models to improve reducing the required communication cost for federated learning while maintaining the accuracy of a global model. Updating the parameters in other neural network models should be observed during the local training procedure on each local edge devices for reducing communication cost efficiently. In addition, larger number of edge devices should be used to evaluate the performance of the proposed method.

5. LiberatAI Infrastructure

This section explains the proposed infrastructure to enable the collaborative development of machine learning models on heterogeneous environments while preserving data privacy.

5.1 System Architecture

LiberatAI enables the collaborative development of machine learning models on heterogeneous environments while preserving data privacy. Users can train the model with their local dataset on their environments and share the trained model via LiberatAI for aggregating the model with others from other developers or researchers because each model requires different hardware and software resources for training or inference. Federated learning is applied to train the models collaboratively by keeping the dataset on the client side to protect data exploitation. The model is exchanged between a server and clients instead of exchanging the raw data. Since edge devices have more computing power than ever, I can deploy and train the model on the client side. However, each environment has its own limitation so LiberatAI provides the model and execution environment which is compatible with the existing client's hardware and software environments to avoid the problem of insufficient and incompatible resources.

Training models on the client side is not trivial because the limitation of client hardware needs to be considered. LiberatAI is composed of four main modules: (1) compressor module for reducing the model size to fit in heterogeneous storage capacity to handle heterogeneous storage resources by applying the proposed method in chapter 2, (2) aggregator module for aggregating the heterogeneous trained models from diverse computing resources to handle heterogeneous computing resources by applying the proposed method in chapter 3, and (3) sparsifier module for saving the communication cost when the models are exchanged between a server and the clients to handle heterogeneous communication resources by applying the proposed method in chapter 4

Figure 39 presents the workflow of LiberatAI.

1. User selects a model in the model repository.

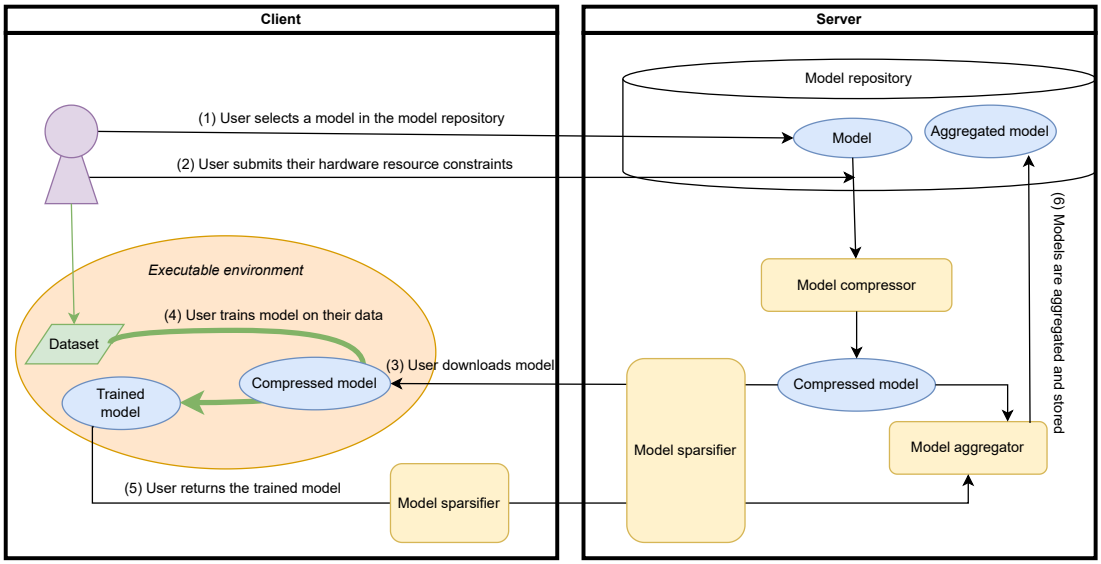


Figure 39: Workflow of LiberatAI

2. User submits the hardware resource constraints of their client device.
3. User downloads the machine learning model which is compatible for their hardware resource.
4. User uses the model to train on their own dataset in the executable environment
5. After local training, the user uploads the trained model to the server via the model sparsifier.
6. The trained model is stored in the model repository and is aggregated to the existing model.

5.2 Evaluation

This section evaluates LiberatAI when applied to a real-world machine learning application. I measure the accuracy of the aggregated model, the model size, and the communication cost to investigate training the models in diverse environments including heterogeneous storage, computing, and communication

Table 19: Model specification

Name	Size [MB]	MFLOPs
COVID-NET	394.65	198.58
ResNet152	243.19	121.07
ResNet101	179.83	88.39
DenseNet201	82.91	40.13
MobileNet	17.27	8.52

Table 20: Distribution of chest X-ray images

Dataset	COVID-19 Negative	COVID-19 Positive	Total
Training	13,992	15,994	29,986
Testing	200	200	400

resources. Lastly, I model and estimate the training time using LiberatAI in large-scale deployments.

This section evaluates LiberatAI when applied to a real-world machine learning application. I measure the accuracy of the aggregated model, the model size, and the communication cost to investigate training the models in diverse environments including heterogeneous storage, computing, and communication resources. Lastly, I model and estimate the training time using LiberatAI in large-scale deployments.

5.2.1 Experimental Setup

LiberatAI was evaluated in a scenario to train models that detect COVID-19 from chest X-ray images. COVID-19 detection is a typical privacy-sensitive use case of machine learning because chest X-ray images may be used to identify patients and could lead to privacy violations [118]. I used COVID-NET, a deep convolutional neural network model tailored for the detection of COVID-19 [119]. In addition, I selected four other popular image classification models: ResNet152, ResNet101, DenseNet201, and MobileNet. Table 19 summarizes the specifications of the models. As for the chest X-ray images to train the models, I used the

Table 21: Hardware specifications of each device

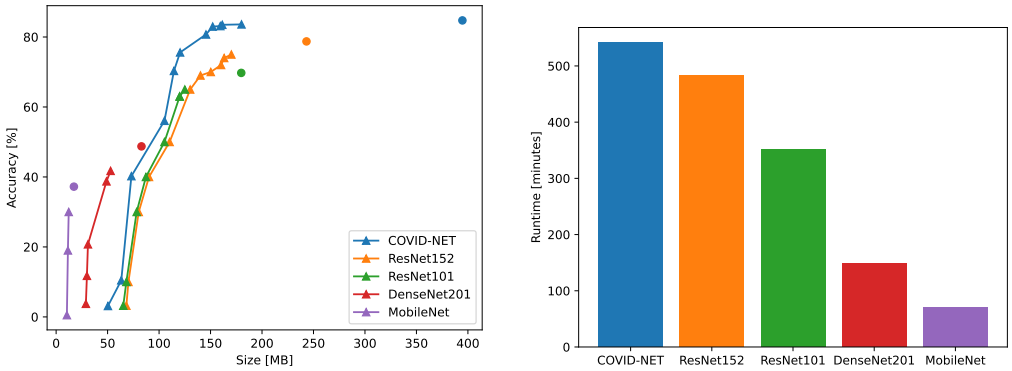
#	**Hardware**	**Specification**
1	GPU	NVIDIA A100 40 GB
	CPU	Intel Xeon Gold 6230R × 2
	Memory	256 GB
2	GPU	NVIDIA RTX 3090 Ti
	CPU	Intel Core i9-12900K × 1
	Memory	64 GB
3	GPU	NVIDIA RTX 3090
	CPU	Intel Core i9-9900K × 1
	Memory	32 GB
4	CPU	Intel Xeon Gold 6230R × 2
	Memory	384 GB
5	CPU	Apple M1 Max
	Memory	32 GB
6	CPU	ARM Cortex-A72 × 1
	Memory	8 GB

COVIDx dataset, the largest open-access benchmark dataset in terms of the number of COVID-19 positive patient cases [120]. I divided the COVIDx dataset into training and testing datasets. The distribution of chest X-ray images is shown in Tab. 20.

Table 21 presents the devices used to evaluate LiberatAI. Each device represents a different class of computing hardware. Device #1 represents a GPU-equipped server, #2 and #3 represent a GPU-equipped desktop PC, #4 represents a CPU-only server, #5 represents a laptop PC, and #6 represents a mobile device. I use these various hardware resources for testing the feasibility of collaboratively developing models on heterogeneous environments using LiberatAI.

5.2.2 Results for Heterogeneity of Storage Resource

The compressor module is evaluated with five image classification models for COVID-19 detection from chest X-ray images. The five models are trained and tested over the same training and testing datasets. Each image classification model has a different model size and model accuracy. In a real-world application, I cannot deploy large models on limited-resource edge devices. Thus, I have to compress them for storing the models based on the storage capacity.



(a) Pareto front for each machine learning model

(b) Runtime for compressing each machine learning model

Figure 40: Pareto front and runtime for compressing each machine learning model

Figure 40a shows the objective space between model size and model accuracy for each machine learning model when the circles represent the original model and the triangles represent the optimal compressed model on the Pareto front. When the accuracy loss of the compressed model is less than 1%, I selected the smallest compressed model to calculate the compression ratio from the compressor module. The compressor module reduced the model size of COVID-NET, ResNet152, ResNet101, DenseNet201, and MobileNet by 54.37%, 36.18%, 30.48%, 30.02%, and 29.18%, respectively. The figure indicates that larger classification models are more amenable to compression than smaller models. MobileNet is compressed with the smallest compression ratio. This might stem from the fact that MobileNet is the smallest model among the classification models, and the fraction of necessary parameters is larger than in other models.

I measured the runtime for compressing each machine learning model as shown

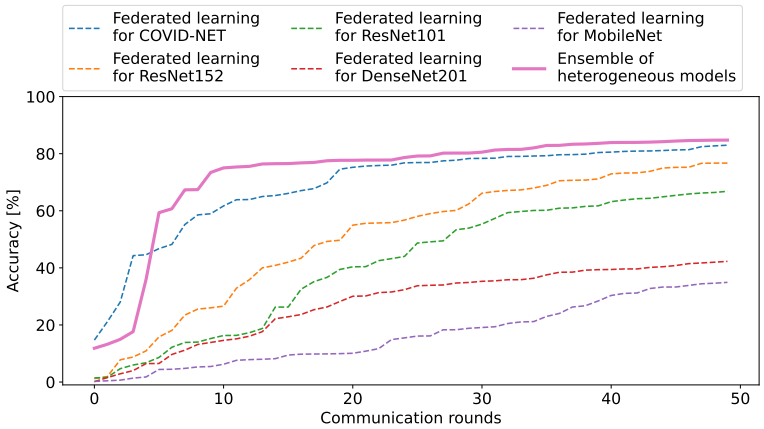


Figure 41: Accuracy of global model for heterogeneous federated learning

in Fig. 40b. Finding the optimized Pareto front for compressing COVID-NET takes the longest runtime around 541 minutes. I found out that a smaller model takes a shorter runtime due to the model size and complexity.

5.2.3 Results for Heterogeneity of Computing Resource

The aggregator module in LiberatAI is evaluated by aggregating the five image classification models to observe the accuracy of aggregated model. Each image classification model is deployed to 20 clients so that the total number of clients is 100 clients. The training dataset of the COVIDx dataset is equally distributed over every client. The testing dataset of the COVIDx dataset is stored on a centralized server to evaluate the aggregated model at each communication round.

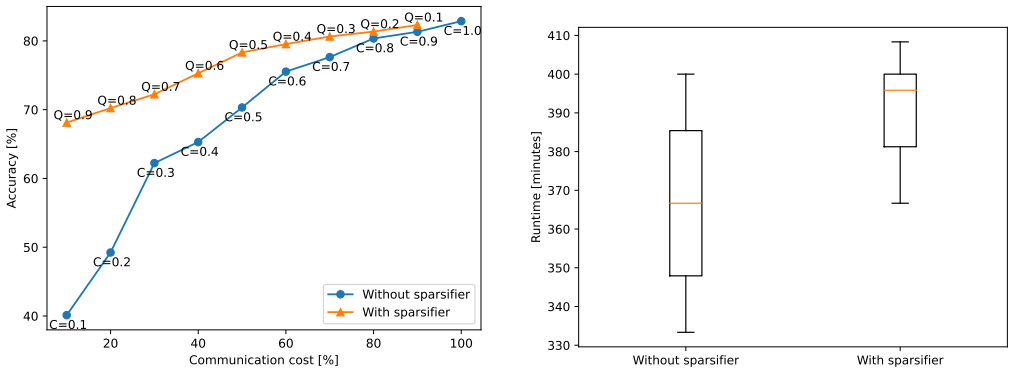
Figure 41 presents the global model accuracy of federated learning when the models with heterogeneous structures are deployed on heterogeneous devices. The accuracy of the global model varies depending on the size and complexity of each model. COVID-NET produces the highest global model accuracy when applying homogeneous federated learning to train the models. From an aggregator module, the ensemble model produces higher accuracy than homogeneous federated learning of COVID-NET since the fifth communication round.

Moreover, I investigate the runtime overhead required to aggregate the heterogeneous models. The runtime with and without the aggregator module is 381 and 415 minutes, respectively. Hence, the aggregator module added a runtime

overhead of 8.19%.

5.2.4 Results for Heterogeneity of Communication Resource

The sparsifier module is evaluated by the communication cost between the centralized server and clients. Since I focus on the limitation of communication resources, I only select the largest model, i.e., COVID-NET, to observe the loss of model accuracy and communication cost. COVID-NET is distributed over 100 clients and each client has an equally distributed training dataset of the COVIDx dataset. I use the testing dataset of COVIDx dataset to evaluate the global model in each communication round.



(a) Accuracy of global model and communication cost

(b) Runtime of with and without the sparsifier module

Figure 42: Accuracy and runtime of with and without the sparsifier module

The sparsifier module has the Q parameter that allows adjusting the quantile ratio of the non-updated parameters that are omitted. On the other hand, communication without the sparsifier module has C parameter in the traditional federated learning algorithm like FedAvg to adjust the fraction of participated clients in each communication round. Figure 42a presents the global model accuracy and total communication cost without and with the sparsifier module in LiberatAI. In each communication round, the communication cost is calculated by the size of the transferred model on both the uplink and downlink directions. When varying the parameter to adjust the model accuracy and communication cost, the result indicates that communication with the sparsifier module is able

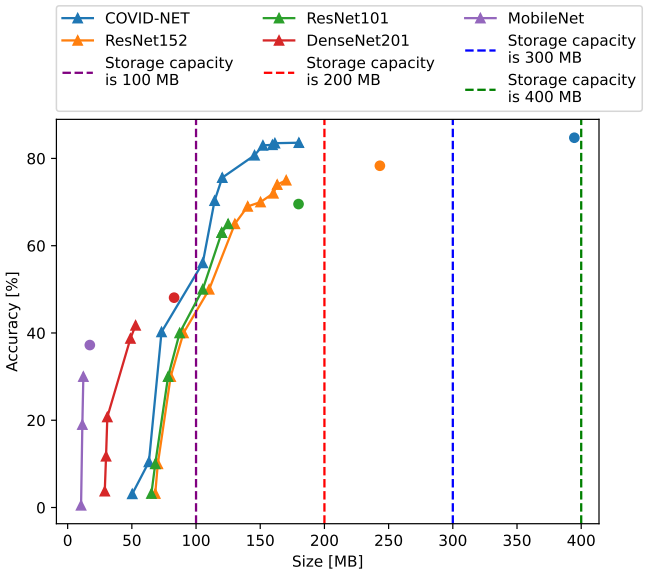


Figure 43: Pareto front for each model with storage capacity constraints

to maintain higher accuracy and lower communication cost than communication without the sparsifier module.

I also measured the time for communication both without and with the sparsifier module as shown in Fig 42b when Q and C are equal to 0.9 and 0.1, respectively. On average, in every communication round, communication with the sparsifier module is added overhead around 6.13% from communication without the sparsifier module. The overhead runtime of the sparsifier module occurs from finding the updated parameters on the client side and sparsifying the model on both the server and client sides.

5.2.5 Results for Heterogeneity of Storage and Computing Resources

I evaluate the compressor and aggregator modules at the same time to investigate if the storage and computing constraints can be simultaneously satisfied. I used five image classification models, COVID-NET, ResNet152, ResNet101, DenseNet201, and MobileNet where each model architecture is distributed to 20 clients. I set four different storage capacity constraints (100 MB, 200 MB, 300 MB, and 400 MB). Figure 43 presents the Pareto front for each machine learning model with different storage capacity constraints.

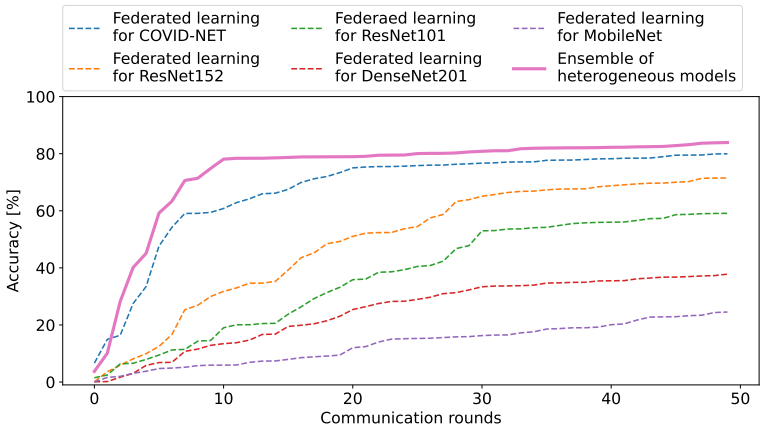


Figure 44: Accuracy of a global model for heterogeneous federated learning

I select the configuration that produces the highest accuracy and fits into the storage capacity constraint. DenseNet201 and MobileNet can be sent to all clients without compression since their model size is smaller than 100 MB. ResNet101 is sent to 15 clients because its size is smaller than 200 MB but larger than 100 MB. For the remaining 5 clients, ResNet101 is compressed to 87.23 MB before being sent. COVID-NET and ResNet152 are sent to 10 clients since their sizes are between 200 MB and 400 MB. However, for the remaining 10 clients, COVID-NET is compressed to either 180.08 MB or 73.12 MB, depending on their storage capacity, and ResNet152 is compressed to either 170.23 MB or 90.23 MB, again depending on their storage capacity.

As a result of the distribution among original and compressed models based on storage capacity constraints, I applied the aggregator module to aggregate the models from heterogeneous clients. Figure 44 indicates that the aggregator module is able to produce higher model accuracy than homogeneous federated learning.

5.2.6 Results for Heterogeneity of Storage and Communication Resources

I evaluate the compressor and sparsifier modules at the same time to investigate if the storage and communication constraints can be simultaneously satisfied. I used the largest image classification for COVID-19 detection like COVID-NET

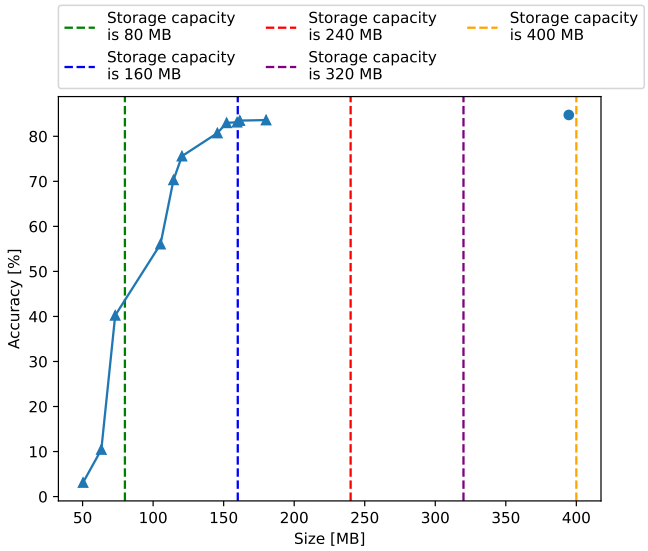


Figure 45: Pareto front for COVID-NET with storage capacity constraints

which is distributed to over 100 clients. I set five different storage capacities (80 MB, 160 MB, 240 MB, 320 MB, and 400 MB) so there are 20 clients for each storage capacity. For each storage capacity, two clients each were set with a quantization ratio from 0.0 to 0.9. Figure 45 presents the Pareto front for COVID-NET with different storage capacity constraints.

I select the configuration that produces the highest accuracy and fits into the storage capacity by the compressor module as follows. The original COVID-NET without any compression is distributed to 20 clients with 400 MB of storage capacity. For deployment to 40 clients with 240 MB or 320 MB of storage capacity, 20 clients with 160 MB of storage capacity, and 20 clients with 80 MB of storage capacity, the model would be compressed to 180.08 MB, 159.51 MB, and 73.12 MB, respectively.

As a result of the distribution among original and compressed models based on storage capacity constraints, I applied the sparsifier module to sparsify model from heterogeneous communication and storage resources. Figure 46 indicates applying the compressor and sparsifier modules are able to execute federated learning on heterogeneous communication and storage resources with maintaining global model accuracy and less communication cost and storage capacity.

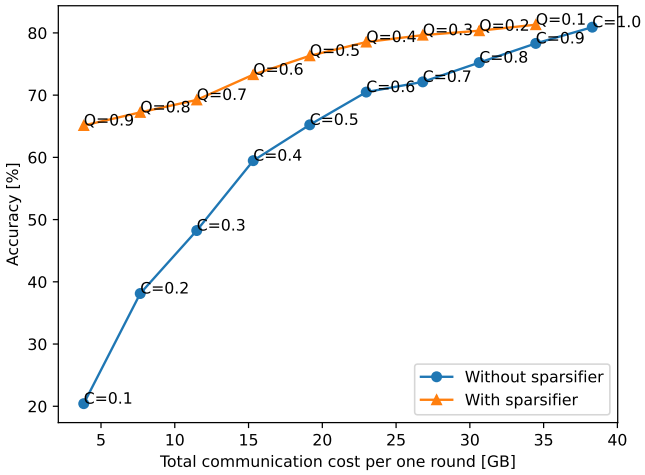


Figure 46: Accuracy of global model and total communication cost

Runtime for applying both compressor and sparsifier is measured to evaluate the overhead added from LiberatAI. Finding the optimized configurations for compressing COVID-NET takes around 541 minutes. Figure 47 indicates that the runtime of the sparsifier module is increased when the number of quantile ration is increased because it needs computing time to calculate more sparsifier models than denser ones.

5.2.7 Results for Heterogeneity of Computing and Communication Resources

I evaluate the aggregator and sparsifier modules at the same time to investigate if the computing and communication constraints can be simultaneously satisfied. I used five image classification models: COVID-NET, ResNet152, ResNet101, DenseNet201, and MobileNet. Each model architecture is distributed to 20 clients. For each model architecture, there are two clients for each quantization ratio from 0.0 to 0.9.

Figure 48a presents the accuracy of global models without and with aggregator and sparsifier modules. Using the aggregator module to combine heterogeneous models improves the model accuracy to be higher than without the aggregator module by 1.78%. Interestingly, the sparsifier module maintains the model accuracy of the aggregated model from the aggregator module when Q is less than 0.3.

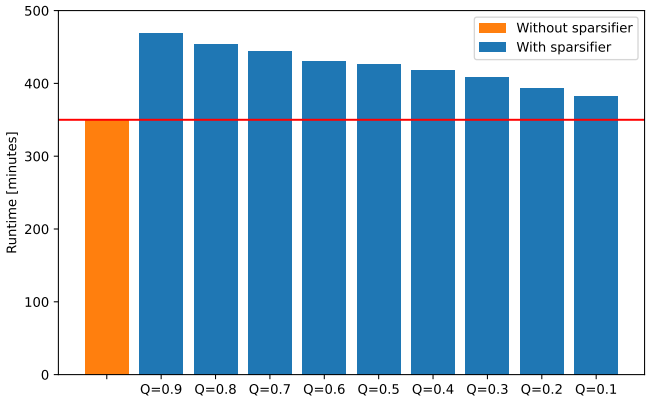


Figure 47: Runtime without and with sparsifier module for different Q

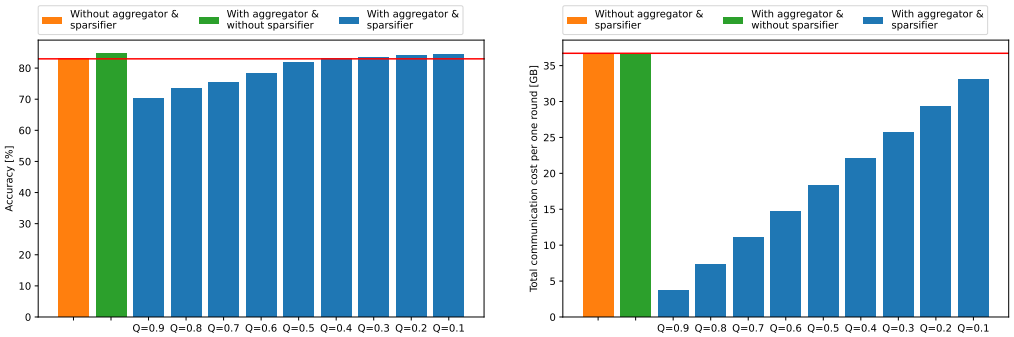
Therefore, applying the aggregator and sparsifier modules produces the highest accuracy and reasonable communication cost when Q is equal to 0.2 as shown in Fig. 48b. The total communication cost in one communication round is calculated from both uplink and downlink directions.

Runtime for applying the aggregator and sparsifier modules is measured to compute the overhead from the proposed modules as shown in Fig. 48c. Aggregating the model with heterogeneous structures using the aggregator module added an overhead runtime of around 7.13%. For the overhead of the sparsifier module, the maximum and minimum overhead runtime is added to the conventional method by 21.97% and 9.74% when Q is equal to 0.9 and 0.1, respectively.

5.2.8 Results for Heterogeneity of Storage, Computing, and Communication Resources

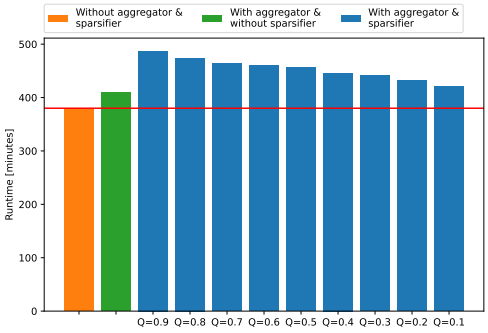
I evaluate compressor, aggregator, and sparsifier modules at the same time to investigate if the storage, computing, and communication constraints can be simultaneously satisfied. I used five classification models, COVID-NET, ResNet152, ResNet101, DenseNet201, and MobileNet, and 20 clients were assigned to each model architecture with a combination of quantization ratios of 0.0 to 0.9 and storage capacities of 400 MB and 200 MB.

Figure 45 presents the Pareto front for each model with different storage capacity constraints. I select the configuration that produces the highest accuracy



(a) Model accuracy

(b) Communication cost



(c) Runtime

Figure 48: Result for applying aggregator and sparsifier modules

and is fits in the storage capacity from the compressor module. ResNet101, DenseNet201, and MobileNet are distributed to all 20 clients without compression since their model size is below 200 MB. Similarly, COVID-NET and ResNet152 are also sent to 10 clients without any compression since their model size is less than 400 MB. However, for an additional 10 clients, COVID-NET is compressed to 180.08 MB to fit the storage capacity of 200 MB, while ResNet152 is compressed to 170.23 MB for these same 10 clients to meet the storage capacity constraints.

As a result of the distribution among original and compressed models based on storage capacity constraints, I applied the sparsifier module to reduce the communication cost between a server and clients. Figure 50 indicates the model accuracy of different total communication costs per one communication round when applying aggregator and sparsifier modules. It is obvious that applying the sparsifier module can maintain the model accuracy better with less communi-

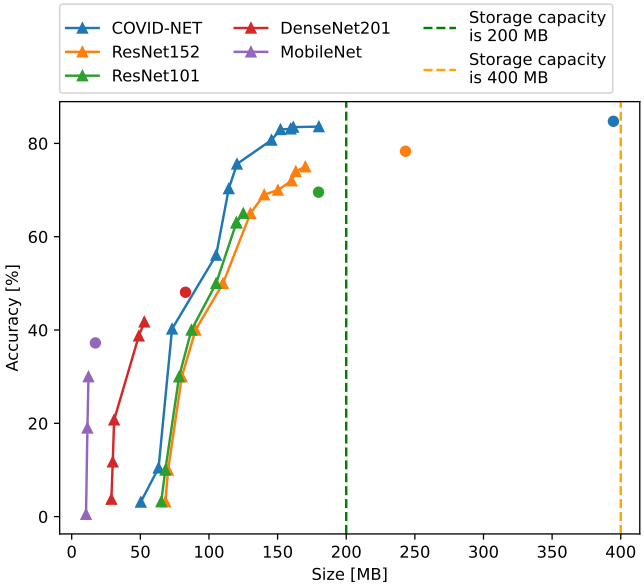


Figure 49: Pareto front for each model with storage capacity constraints

cation cost better than without the sparsifier module. On average for applying the aggregator module, I found out that model accuracy is increased by 2.8% and 1.6% without and with the sparsifier module, respectively. Afterward, the aggregator module is used to aggregate the model with heterogeneous structures. The result indicates applying the proposed model aggregator module is able to enhance model accuracy. However, in the current evaluation of COVID-19 detection, the evaluation dataset has limited data diversity, resulting in only a slight improvement.

I measured runtime for applying the sparsifier module without and with the aggregator module as shown in Fig. 51a and Fig. 51b, respectively. On average, the runtime is added around 1.18% and 1.54% for without and with the sparsifier module.

5.2.9 Runtime Analysis

Apart from simultaneously satisfying the storage, computing, and communication constraints on heterogeneous environments, the runtime is also an important factor to evaluate the proposed infrastructure. Given the limitation in available

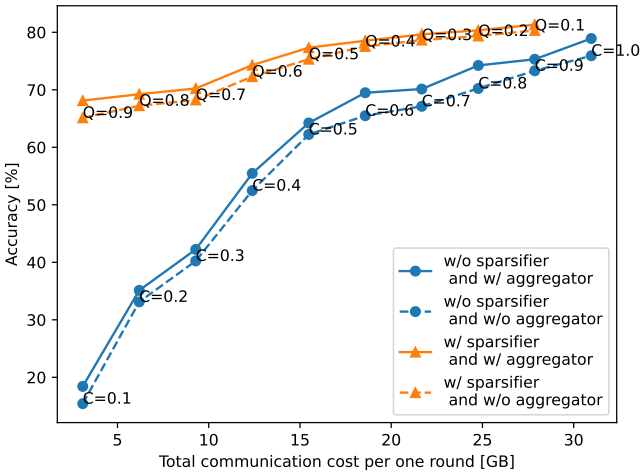
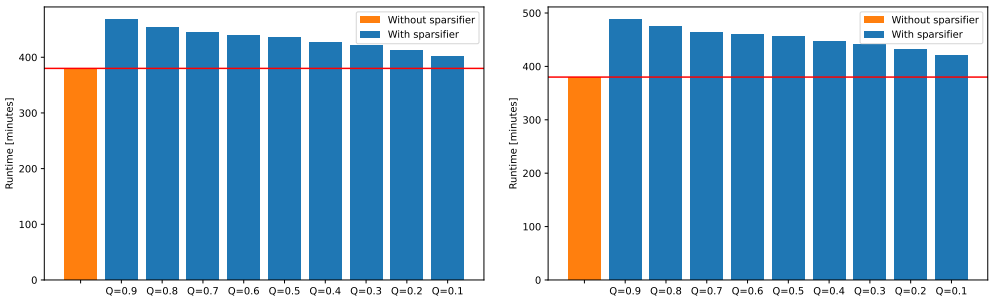


Figure 50: Communication cost when applying aggregator and sparsifier



(a) Without aggregator module (b) With aggregator module

Figure 51: Runtime for applying sparsifier

resources, I model the training time and estimate the training time in large-scale heterogeneous environments.

I first model the time for training on the client side. I use actual measurements to estimate the local training time as shown in Fig. 52. With respect to the communication time, I assume that only C clients can communicate with the server at the same time, and the throughput of uploading or downloading a model is constant. Communication and local training are performed in a pipelined manner as depicted in Fig. 53. For example, once the first group of C clients completes downloading the global model from the server, the first group starts the local training and the next group starts downloading the global model. As a

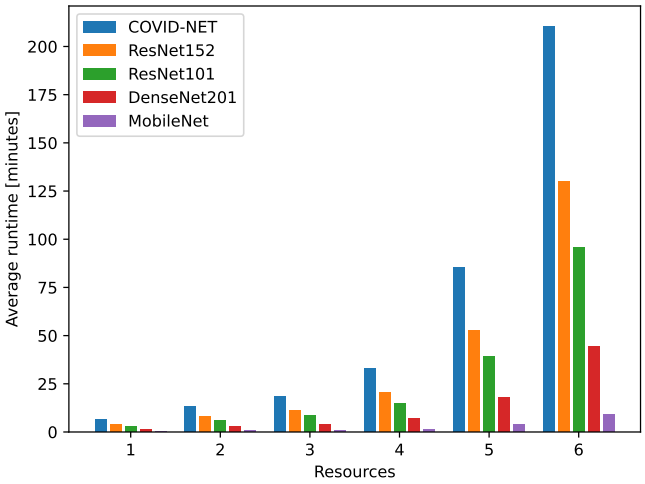


Figure 52: Training time of models on different devices

result, the total training time for a single communication round is modeled as

$$T_{train} = \left(\sum_{n=0}^{\lceil (N/C)-1 \rceil} \max_{i=nC}^{[nC+C]} \frac{S_M^i}{B} \right) + \max_{i=N-C}^{N} t_r^{mi} + \frac{S_M^i}{B}, \tag{3}$$

where N is the total number of clients, B is the throughput of communication between the server and a client, S_M^i is the size of the model and t_r^{mi} the time for local training.

On the server side, the time for aggregating and calculating the ensemble of models must be considered. The total runtime of a single communication round for training homogeneous models is modeled as

$$T_{homo} = T_{train} + aN, \tag{4}$$

where a is the time required for aggregation per client. The total runtime of a single communication round for training heterogeneous models is modeled as

$$T_{hetero} = T_{homo} + eM, \tag{5}$$

where e is the ensemble time per model architecture and M is the total number of model architectures. Based on preliminary experiments, a is set to 1.14s and e is set to 3.28s in this evaluation.

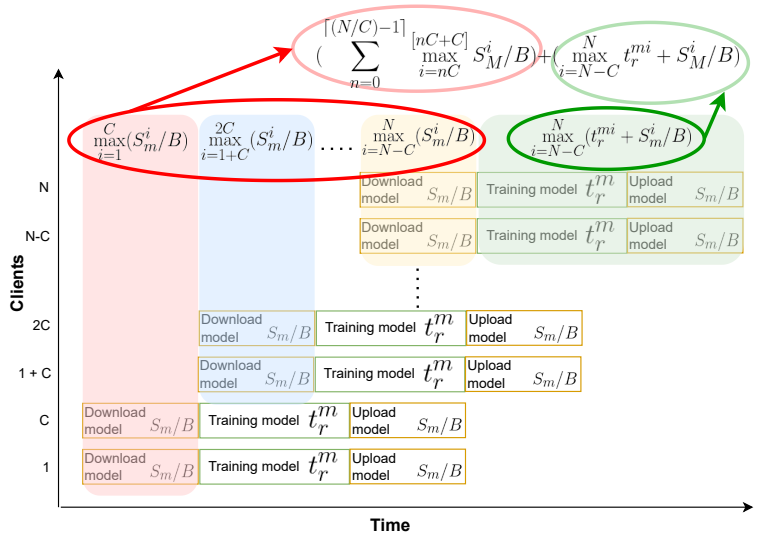
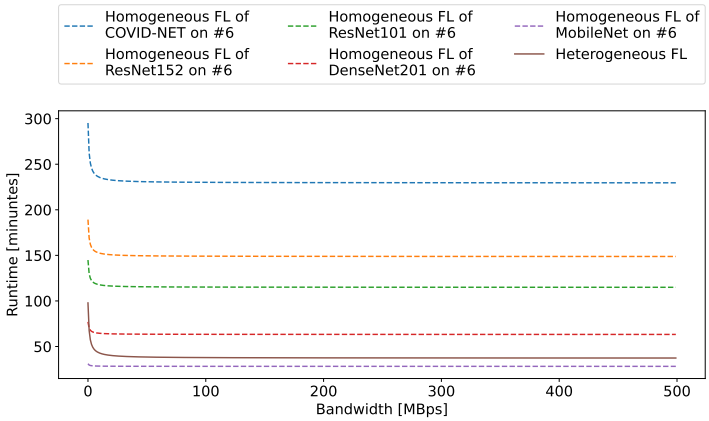


Figure 53: Runtime diagram for local training in federated learning

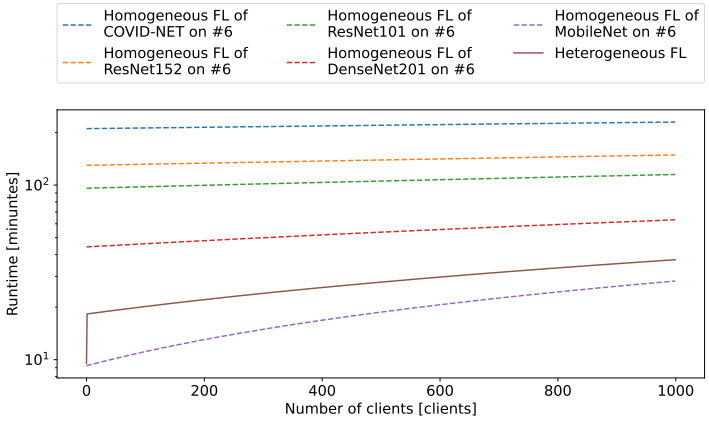
I estimate the runtime for homogeneous federated learning when the model is deployed on the first and sixth environments since the first and sixth devices are the fastest and slowest devices. For heterogeneous federated learning, COVID-NET is deployed on the first device, ResNet152 is deployed on the second device, ResNet101 is deployed on the third device, DenseNet201 is deployed on the fifth device and MobileNet is deployed on the sixth device. I distributed each model architecture on a different device balancing the runtime across various devices.

Figure 54a shows the runtime when varying bandwidth from 1 Mbps to 500 Mbps. The number of clients is fixed to 1,000 and the number of concurrent connections is 100. The runtime decreases sharply with the bandwidth up to 20 Mbps because the communication time (the first term in Equation (3)) is the bottleneck . The bandwidth has no significant impact on the runtime when the bandwidth is higher than 20 MBps.

Additionally, I varied the number of clients as shown in Fig. 54b. Here, the bandwidth is 500 Mbps and the maximum number of connections is 100. When the number of clients is 1,000, the runtime of heterogeneous federated learning is slower than that of homogeneous federated learning of MobileNet by 24.47%, but is faster than that of homogeneous federated learning of COVID-NET, ResNet152, ResNet101, and DenseNet201 by 83.73%, 74.89%, 67.51%, and 40.93%,



(a) Varying the network bandwidth



(b) Varying the number of clients

Figure 54: Estimated training time

respectively. However, heterogeneous federated learning achieved higher model accuracy than homogeneous federated learning by 1.78% as shown in Fig. 41. Therefore, LiberatAI enables heterogeneous federated learning to achieve higher model accuracy and take less runtime than homogeneous federated learning of COVID-NET, ResNet152, ResNet101, and DenseNet201.

5.3 Conclusion and Future Work

This chapter proposed LiberatAI which is a federated infrastructure to enable the collaborative development of machine learning models on heterogeneous environ-

ments. LiberatAI allows individuals to participate in collaborative development by training the models on their environments which usually be heterogeneous. There are three modules were proposed in LiberatAI to support training a model on diverse storage, computing, and communication resources. (1) compressor module was proposed to reduce the model size to fit in the storage capacity of the environment. (2) aggregator module was proposed to aggregate the models with heterogeneous on heterogeneous computing resources. (3) sparsifier module was proposed to sparsify the model for exchanging the model between a server and clients.

LiberatAI was evaluated using state-of-the-art neural network models for the detection of COVID-19 cases from chest X-ray images. COVID-19 detection is one of the most popular machine learning applications to apply a machine learning model on privacy-sensitive data. I trained COVID-NET over six heterogeneous environments while preserving data privacy. As a result, LiberatAI allows collaborative development to develop a machine learning model for detecting COVID-19 on diverse environments. LiberatAI has an aggregator module to enhance prediction accuracy by aggregating heterogeneous machine learning models from heterogeneous environments and a sparsifier module to reduce communication costs between a server and clients while maintaining the model accuracy. The compressor module in LiberatAI finds the configuration automatically for compressing the model to fit in diverse storage capacities with comparable accuracy. Additionally, I conduct the runtime estimation to calculate the runtime for homogeneous and heterogeneous federated learning based on LiberatAI.

In the future, the generality of LiberatAI will be investigated using a variety of machine learning applications with diverse structures of machine learning models. I plan to evaluate LiberatAI on a large number of edge devices and then improve the resource utilization in the infrastructure. I will make LiberatAI to be compatible with all machine learning libraries because the current compatibility of LiberatAI with other libraries is still limited to Scikit-learn, Tensorflow, Keras, Theano, and PyTorch libraries.

Federated learning preserves data privacy by keeping the training data on the devices of the users, and not sharing the data with a central server. However, there are still challenges such as ensuring that the data is properly encrypted

may be affected by the privacy-preserving techniques used. Overall, federated learning is a promising approach for preserving data privacy in machine learning but requires careful consideration and management of the system to ensure that data remains secure while allowing for effective model training.

Additionally, I will make LiberatAI as open-source software and available for the international or domestic research communities to remove the barrier to the collaborative development of machine learning models from the limitation of data privacy and existing resource constraints.

6. Conclusion

6.1 Summary

In this dissertation, I proposes *LiberatAI*, an infrastructure that enables the collaborative development of machine learning models on heterogeneous environments while preserving data privacy. Federated learning approach is applied to train the models without exchanging the raw dataset between a server and clients. LiberatAI allows individuals to participate in collaborative development by training the models on their environments which usually be heterogeneous. There are three modules were proposed in LiberatAI to support training a model on diverse storage, computing, and communication resources.

To support training a model on heterogeneous storage resources, I proposed a method for reducing the model size to fit in heterogeneous storage capacity while maintaining the original model accuracy. To balance the trade-off between model size and accuracy, conventional model compression methods require manual effort to find the optimal configuration that reduces the model size without significant degradation of accuracy. The proposed method is automatically finding the optimal configurations for quantization. The proposed method suggests multiple compression configurations that produce models with different sizes and accuracy, from which users can select the configurations that suit their use cases. Additionally, I propose a retraining method that does not require any labeled datasets for retraining. I evaluated the proposed method using various neural network models for classification, regression, and semantic similarity tasks and demonstrated that the proposed method reduced the size of models by at least 30% while maintaining less than 1% loss of accuracy. I compared the proposed method with state-of-the-art automated compression methods and showed that it can provide better compression configurations than existing methods.

To support training a model on heterogeneous computing resources, I proposed a method for aggregating the heterogeneous trained models from diverse computing resources. Existing federated learning algorithms assume that all deployed models share the same structure. However, it is often infeasible to distribute the same model to every edge device because of hardware limitations such as computing performance. I propose a novel federated learning algorithm to ag-

gregate information from multiple heterogeneous models. The proposed method uses a weighted average ensemble to combine the outputs from each model. The weight for the ensemble is optimized using black-box optimization methods. I evaluated the proposed method using diverse models and datasets and found that it can achieve comparable performance to conventional training using centralized datasets. Moreover, I compared six different optimization methods to tune the weights for the weighted average ensemble and found that tree parzen estimator achieves the highest accuracy among the alternatives.

To support training a model on heterogeneous network resources, I proposed a method for saving communication costs when the models are exchanged between a server and the clients. The proposed method transfers only top-updated parameters in neural network models to reduce the required communication cost for federated learning. The proposed method allows adjusting the criteria of updated parameters to trade off the reduction of communication costs and the loss of model accuracy. I evaluated the proposed method using diverse models and datasets and found that it can achieve comparable performance to transfer original models for federated learning. As a result, the proposed method has achieved a reduction of the required communication costs by around 90% when compared to the conventional method for VGG16. Furthermore, I found out that the proposed method is able to reduce the communication cost of a large model more than of a small model due to the different thresholds of updated parameters in each model architecture.

Finally, I integrate my proposed methods to build LiberatAI infrastructure. LiberatAI was evaluated using state-of-the-art neural network models for the detection of COVID-19 cases from chest X-ray images. COVID-19 detection is one of the most popular machine learning applications to apply a machine learning model on privacy-sensitive data. I trained COVID-NET over six heterogeneous environments while preserving data privacy. As a result, LiberatAI allows collaborative development to develop a machine learning model for detecting COVID-19 on diverse environments. LiberatAI has an aggregator module to enhance prediction accuracy by aggregating heterogeneous machine learning models from heterogeneous environments and a sparsifier module to reduce communication costs between a server and clients while maintaining the model accuracy. The

compressor module in LiberatAI finds the configuration automatically for compressing the model to fit in diverse storage capacities with comparable accuracy. Additionally, I conduct the runtime estimation to calculate the runtime for homogeneous and heterogeneous federated learning based on LiberatAI.

From the result of this dissertation, LiberatAI shows a potential to remove the barrier for the collaborative development of machine learning models from the limitation of data privacy and heterogeneous environments. Many machine learning models will be built to support multidisciplinary research since researchers are able to contribute the existing models without training the models from scratch which requires a significant amount of computing resources. LiberatAI will allow machine learning developers or researchers to develop machine learning models collaboratively. Research communities in both academia and industry will be expanded and crossed over multidisciplinary because of the infrastructure. The number of research collaborations will be continuously increased because the barrier of data usage and hardware resources has already been eliminated. LiberatAI might enable emerging models in various research fields, especially the fields that utilize privacy-sensitive data when it is available. Furthermore, LiberatAI will attract many researchers to build research communities by sharing their knowledge and experience with each other.

6.2 Future Work

In the future, the generality of LiberatAI will be investigated using a variety of machine learning applications with diverse structures of machine learning models. I plan to evaluate LiberatAI on a large number of edge devices and then improve the resource utilization in the infrastructure. Additionally, I will make LiberatAI as open-source software and available for the international or domestic research communities to remove the barrier to the collaborative development of machine learning models from the limitation of data privacy and existing resource constraints.

Acknowledgements

I would like to thank the following people for their wisdom, guidance, and support. Without their help, this work would never have been possible.

First and foremost, I would like to express my gratitude to Professor Hajimu Iida for providing a great research environment. His laboratory, Laboratory for Software Design and Analysis, is a great place to pursue research.

To Professor Kazutoshi Fujikawa, I appreciate for his constructive comments and feedback made my work come this far. Without him, my research work and the dissertation would not have been possible.

I would like to express deep appreciation to my supervisors, Associate Professor Kohei Ichikawa and Assistant Professor Keichi Takahashi for their continuous support and guidance in my research work as well as my life in Japan. Their valuable suggestions and comments brought this research to fruition. Without them, I would not have successfully accomplished the doctoral course.

To Assistant Professor Putchong Uthayopas, who was also my advisor during my time as an undergraduate student at Kasetsart University. He gave me invaluable knowledge in research methodology and widened my vision in the area of high-performance computing. His insightful suggestion helped shape this research in its initial stage. Without him, I could not come this far. Deep in my mind, I will always keep his image and he will always be remembered forever.

To Assistant Professor Chawanat Nakasan, I appreciate for his informative feedback and suggestions always helps raise the quality of this research.

I would like to acknowledge my dissertation committee. Thank you so much for reviewing my dissertation and for the insightful comments and suggestions that helped me to improve the overall quality of this dissertation.

To PRAGMA, Dr. Peter Arzberger, Ms. Shava Smallen, and Ms. Nadya Williams, I appreciate for providing me with a lot of assistance, and advice in organizing PRAGMA workshops and mentoring PRAGMA Students. I also express my gratitude towards Dr. Jason Haga and Dr. Prapaporn Rattanatamrong for their guidance to PRAGMA Students Steering Committee.

To Friendship, I would like to express my thanks to all of my Thai and international friends. I feel joyful every time when we stay and travel together. Without them, my daily life could not be enjoyed like this.

Last but not least, I wish to express my highest gratitude to my dearest family for the support, raising, and educating me with great care since my youth. No amount of words would sufficiently express my gratitude.

Finally, I would like to thank the Ministry of Education, Culture, Sports, Science and Technology (MEXT) Scholarship and the Japan Society for the Promotion of Science (JSPS) DC2 Research Fellowship for the monetary support. This scholarship enables me to live and pursue my research in Japan comfortably. It is a huge honor to be a recipient of this scholarship.

References

[1] Soni Singh, K R Ramkumar, and Ashima Kukkar. Machine learning techniques and implementation of different ml algorithms. In *2021 2nd Global Conference for Advancement in Technology (GCAT)*, pages 1–6, 2021.

[2] Gregory Moro Puppi Wanderley, Marie-Hélène Abel, Jean-Paul Barthès, and Emerson Cabrera Paraiso. An advanced collaborative environment for software development. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 002917–002922, 2016.

[3] Carolyn Wong. A successful software development. *IEEE Transactions on Software Engineering*, SE-10(6):714–727, 1984.

[4] Ya-Wen Yu, Yu-Shing Chang, Yu-Fu Chen, and Li-Sheng Chu. Entrepreneurial success for high-tech start-ups – case study of taiwan high-tech companies. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 933–937, 2012.

[5] Karthik Navuluri, Ravi Mukkamala, and Aftab Ahmad. Privacy-aware big data warehouse architecture. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 341–344, 2016.

[6] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, pages 1–10, 2016.

[7] Jeronimo Castrillon, Matthias Lieber, Sascha Klüppelholz, Marcus Völp, Nils Asmussen, Uwe Aßmann, Franz Baader, Christel Baier, Gerhard Fettweis, Jochen Fröhlich, Andrés Goens, Sebastian Haas, Dirk Habich, Hermann Härtig, Mattis Hasler, Immo Huismann, Tomas Karnagel, Sven Karol, Akash Kumar, Wolfgang Lehner, Linda Leuschner, Siqi Ling, Steffen Märcker, Christian Menard, Johannes Mey, Wolfgang Nagel, Benedikt Nöthen, Rafael Peñaloza, Michael Raitza, Jörg Stiller, Annett Ungethüm,

Axel Voigt, and Sascha Wunderlich. A hardware/software stack for heterogeneous systems. *IEEE Transactions on Multi-Scale Computing Systems*, 4(3):243–259, 2018.

[8] Kundjanasith Thonglek, Keichi Takahashi, Kohei Ichikawa, Chawanat Nakasan, Hidemoto Nakada, Ryousei Takano, and Hajimu Iida. Retraining quantized neural network models with unlabeled data. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2020.

[9] Kundjanasith Thonglek, Keichi Takahashi, Kohei Ichikawa, Hajimu Iida, and Chawanat Nakasan. Federated learning of neural network models with heterogeneous structures. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 735–740, 2020.

[10] Kundjanasith Thonglek, Keichi Takahashi, Kohei Ichikawa, Chawanat Nakasan, Pattara Leelaprute, and Hajimu Iida. Sparse communication for federated learning. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pages 1–8, 2022.

[11] George Plastiras, Maria Terzi, Christos Kyrkou, and Teocharis Theocharidcs. Edge Intelligence: Challenges and opportunities of near-sensor machine learning applications. In *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–7, July 2018.

[12] Farhana Sultana, Abu Sufian, and Paramartha Dutta. Advancements in image classification using convolutional neural network. In *Proceedings of the IEEE International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pages 122–129, November 2018.

[13] Juyong Kim, Yookoon Park, Gunhee Kim, and Sung Ju Hwang. SplitNet: Learning to semantically split deep networks for parameter reduction and model parallelization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1866–1874, August 2017.

[14] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. Predicting the computational cost of deep learning models. In *Proceedings of the IEEE International Conference on Big Data (BigData)*, pages 3873–3882, December 2018.

[15] Yunhe Wang, Chang Xu, Chao Xu, and Dacheng Tao. Beyond Filters: Compact feature map for portable deep model. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3703–3711, August 2017.

[16] Miguel A. Carreira-Perpinan and Yerlan Idelbayev. Learning-Compression algorithms for neural net pruning. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8532–8541, June 2018.

[17] Tara Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6655–6659, May 2013.

[18] Sheng Xu, Anran Huang, Lei Chen, and Baochang Zhang. Convolutional neural network pruning: A survey. In *2020 39th Chinese Control Conference (CCC)*, pages 7458–7463, 2020.

[19] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, pages 2074–2082, December 2016.

[20] Yu Cheng, Quanfu Fan, Sharath Pankanti, and Alok Choudhary. Temporal sequence modeling for video event detection. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2235–2242, June 2014.

[21] Yen-Lin Lee, Pei-Kuei Tsung, and Max Wu. Technology trend of edge

AI. In *Proceedings of the IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–2, April 2018.

[22] Kenyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE Access*, 8:85714–85728, May 2020.

[23] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: A big data - ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1328–1347, 2021.

[24] Yue Zhu, James Kwok, and Zhi-Hua Zhou. Multi-Label learning with global and local label correlation. *IEEE Transactions on Knowledge and Data Engineering*, 30(6):1081–1094, June 2018.

[25] Nimrod Busany, Shahar Maoz, and Yehonatan Yulazari. Size and accuracy in model inference. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 887–898, November 2019.

[26] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *CoRR*, abs/1710.09282, October 2017.

[27] Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. DarkRank: Accelerating deep metric learning via cross sample similarities transfer. In *Proceedings of the Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI)*, pages 2852–2859, October 2018.

[28] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, pages 1–9, March 2015.

[29] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 1–13, September 2014.

[30] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *Proceedings of the International Conference on Learning Representations (ICLR)*, pages 1–13, April 2017.

[31] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of International Conference on Learning Representations*, 2019.

[32] James Diffenderfer and Bhavya Kailkhura. Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network. In *Proceedings of International Conference on Learning Representations*, 2021.

[33] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[34] Xiaotong Lu, Heng Wang, Weisheng Dong, Fangfang Wu, Zhonglong Zheng, and Guangming Shi. Learning a deep vector quantization network for image compression. *IEEE Access*, 7:118815–118825, August 2019.

[35] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, December 2014.

[36] Lei Yang, Shangyou Zeng, Yue Zhou, Bing Pan, Yanyan Feng, and Daihui Li. Design of convolutional neural network based on tree fork module. In *2019 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pages 1–4, 2019.

[37] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. Resiliency of deep neural networks under quantization. *CoRR*, abs/1511.06488, November 2015.

[38] Shangyu Chen, Wenya Wang, and Sinno Pan. Deep neural network quantization via layer-wise optimization using limited training data. In *Proceedings of the Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI)*, pages 3329–3336, July 2019.

[39] He Yihui, Lin Ji, Liu Zhijian, Wang Hanrui, Li Li-Jia, and Han Song. AMC: AutoML for model compression and acceleration on mobile devices. In *Proceedings of European Conference on Computer Vision (ECCV)*, page 1–17, September 2018.

[40] Ahmed Elthakeb, Prannoy Pilligundla, Fatemehsadat Mireshghallah, Amir Yazdanbakhsh, and Hadi Esmaeilzadeh. ReLeQ : A reinforcement learning approach for automatic deep quantization of neural networks. *IEEE Micro*, 40(5):37–45, 2020.

[41] Lou Qian, Guo Feng, Kim Minje, Liu Lantao, and Jiang Lei. AutoQ: Automated kernel-wise neural network quantization. In *Proceedings of International Conference on Learning Representations (ICLR)*, page 1–11, May 2020.

[42] Komal Sharma and Kunal Gupta. Lossless data compression techniques and their performance. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 256–261, 2017.

[43] Mohd Salihin Ngadiman Yusliza Yusoff and Azlan MohdZain. Overview of NSGA-II for optimizing machining process parameters. *Procedia Engineering*, 15:3978–3983, December 2011.

[44] Ke Shang, Hisao Ishibuchi, Linjun He, and Lie Meng Pang. A survey on the hypervolume indicator in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 25(1):1–20, 2021.

[45] Federico Cruciani, Chen Sun, Shuai Zhang, Chris Nugent, Chunping Li, Shaoxu Song, Cheng Cheng, Ian Cleland, and Paul Mccullagh. A public domain dataset for human activity recognition in free-living conditions. In *Proceedings of the IEEE International Conference on SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 166–171, August 2019.

[46] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database, 2010.

[47] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, June 2009.

[48] Alex Krizhevsky. CIFAR-100 dataset, 2010.

[49] David Castro. A 3D version of the MNIST database of handwritten digits, 2019.

[50] Yu Xiang, Wonhui Kim, Wei Chen, Jingwei Ji, Christopher Choy, Hao Su, Roozbeh Mottaghi, Leonidas Guibas, and Silvio Savarese. Objectnet3D: A large scale database for 3D object recognition. In *Proceedings of the European Conference Computer Vision (ECCV)*, October 2016.

[51] Cam Nugent. Historical stock data for all current S and P 500 companies, 2018.

[52] Mark Zielinski. Bitcoin data at 1-min intervals from select exchanges, Jan 2012 to Sept 2020, 2020.

[53] Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. SemEval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 1–8, Dublin, Ireland, August 2014. Association for Computational Linguistics.

[54] Samuel Bowman, Gabor Angeli, Christopher Potts, and Christopher Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[55] Naigong Yu, Panna Jiao, and Yuling Zheng. Handwritten digits recognition base on improved LeNet5. In *Proceedings of the IEEE Chinese Control and Decision Conference (CCDC)*, pages 4871–4875, May 2015.

[56] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[58] Taranjit Kaur and Tapan Kumar Gandhi. Automated brain image classification based on vgg-16 and transfer learning. In *Proceedings of the IEEE International Conference on Information Technology (ICIT)*, pages 94–98, December 2019.

[59] Ching-Kai Tseng, Chien-Chih Liao, Po-Chun Shen, and Jiun-In Guo. Using C3D to detect rear overtaking behavior. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 151–154, September 2019.

[60] Lasani Hussain, Sekhar Banarjee, Sumit Kumar, Aditya Chaubey, and Motahar Reza. Forecasting time series stock data using deep learning technique in a distributed computing environment. In *Proceedings of the IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*, pages 489–493, September 2018.

[61] Zizhuang Wei, Qingtian Zhu, Chen Min, Yisong Chen, and Guoping Wang. Bidirectional hybrid lstm based recurrent neural network for multi-view stereo. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2022.

[62] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[63] Fernando Gonçalves José and G. C. Resende Mauricio. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17:487–525, 2011.

[64] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[65] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, August 2017.

[66] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *CoRR*, 2018.

[67] Mohamed Loey, Ahmed El-Sawy, and Hazem M. El-Bakry. Deep learning autoencoder approach for handwritten arabic digits recognition. *CoRR*, 2017.

[68] Shudong Yang, Xueying Yu, and Ying Zhou. LSTM and GRU neural network performance comparison study: Taking yelp review dataset as an example. In *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, pages 98–101, 2020.

[69] Yang Qiang, Liu Yang, Chen Tianjian, and Tong Yongxin. Federated machine learning: Concept and applications. *ACM Transaction Intelligence System Technology*, 10(2):12–19, January 2019.

[70] Erivaldo Fernandes, Guanci Yang, Manh Do, and Weihua Sheng. Detection of privacy-sensitive situations for social robots in smart homes. In *Proceedings of the IEEE International Conference on Automation Science and Engineering (CASE)*, pages 727–732, August 2016.

[71] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Arcas. Communication-efficient learning of deep networks from decentralize data. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1273–1282, April 2017.

[72] Qi Qi, Qiming Huo, Jingyu Wang, Haifeng Sun, Yufei Cao, and Jianxin Liao. Personalized sketch-based image retrieval by convolutional neural network and deep transfer learning. *IEEE Access*, 7:16537–16549, January 2019.

[73] Chetan Nadiger, Anil Kumar, and Sherine Abdelhak. Federated reinforcement learning for fast personalization. In *Proceedings of the IEEE International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 123–127, June 2019.

[74] Vukasin Felbab, Peter Kiss, and Tomas Horvath. Optimization in federated learning. In *Proceedings of the International Conference on Information Technologies - Application and Theory (ITAT)*, pages 58–65, September 2019.

[75] Anit Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *CoRR*, abs/1812.06127, January 2019.

[76] Johnson Rie and Zhang Tong. Accelerating stochastic gradient descent using predictive variance reduction. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, volume 1, pages 315–323, December 2013.

[77] Gintare Karolina Dziugaite and Daniel Roy. Entropy-SGD optimizes the prior of a pac-bayes bound: Generalization properties of entropy-SGD and data-dependent priors. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1376–1385, February 2018.

[78] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. In *Proceedings of the International Conference on Learning Representations (ICLR)*, pages 1–10, April 2016.

[79] Tian Li, Maziar Sanjabi, and Virginia Smith. Fair resource allocation in federated learning. *CoRR*, abs/1905.10497, May 2019.

[80] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[81] Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, February 2016.

[82] Faliang Huang, Guoqing Xie, and Ruliang Xiao. Research on ensemble learning. In *Proceedings of the International Conference on Artificial Intelligence and Computational Intelligence (AICI)*, volume 3, pages 249–252, January 2009.

[83] Inwoong Lee, Doyoung Kim, Seoungyoon Kang, and Sanghoon Lee. Ensemble deep learning for skeleton-based action recognition using temporal sliding lstm networks. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 1012–1020, 2017.

[84] Li Deng and John Platt. Ensemble deep learning for speech recognition. In *Proceedings of INTERSPEECH*, September 2014.

[85] Feng Xue, Dongliang Wei, Zhi Wang, Tong Li, Yue Hu, and Hongyi Huang. Grid searching method in spherical coordinate for PD location in a substation. In *Proceeding of the International Conference on Condition Monitoring and Diagnosis (CMD)*, pages 1–5, September 2018.

[86] Ying Shang and Jizheng Chu. A method based on random search algorithm for unequal circle packing problem. In *Proceedings of the International Conference on Information Science and Cloud Computing Companion (ISCC-C)*, pages 43–47, December 2013.

[87] Marc Kirschenbaum and Daniel Palmer. Perceptualization of particle swarm optimization. In *Proceedings of the Swarm/Human Blended Intelligence Workshop (SHBI)*, pages 1–5, September 2015.

[88] Alfredo Garcia, Enrique Campos, and Chenyang Li. Distributed on-line Bayesian search. In *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 1–5, December 2005.

[89] Meng Zhao and Jinlong Li. Tuning the hyper-parameters of CMA-ES with tree-structured Parzen estimators. In *Proceedings of the International Conference on Advanced Computational Intelligence (ICACI)*, pages 613–618, March 2018.

[90] Hutter Frank, Hoos Holger, and Leyton Brown Kevin. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*, page 507–523, January 2011.

[91] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. 2011.

[92] Najmul Hassan, Saira Gillani, Ejaz Ahmed, Ibrar Yaqoob, and Muhammad Imran. The role of edge computing in internet of things. *IEEE Communications Magazine*, 56(11):110–115, Nov 2018.

[93] Amir Gandomi and Murtaza Haider. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137–144, Apr 2015.

[94] Yingchun Wang, Jingyi Wang, Weizhan Zhang, Yufeng Zhan, Song Guo, Qinghua Zheng, and Xuanyu Wang. A survey on deploying mobile deep learning applications: A systemic and technical perspective. *Digital Communications and Networks*, Jun 2021.

[95] Zhaoyang Du, Celimuge Wu, Tsutomu Yoshinaga, Kok-Lim Alvin Yau, Yusheng Ji, and Jie Li. Federated learning for vehicular internet of things: Recent advances and open issues. *IEEE Open Journal of the Computer Society*, 1(01):45–61, Jan 2020.

[96] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-i.i.d. data. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3400–3413, Sep 2020.

[97] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, May 2020.

[98] Luping Wang, Wei Wang, and Bo Li. CMFL: Mitigating communication overhead for federated learning. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 954–964, Oct 2019.

[99] Dave Conway-Jones, Tiffany Tuor, Shiqiang Wang, and Kin K. Leung. Demonstration of federated learning in a resource-constrained networked environment. In *Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 484–486, Aug 2019.

[100] Latif Khan, Madyan Alsenwi, Ibrar Yaqoob, Muhammad Imran, Zhu Han, and Choong Seon Hong. Resource optimized federated learning-enabled cognitive internet of things for smart industries. *IEEE Access*, 8:168854–168864, Sep 2020.

[101] Jenny Hamer, Mehryar Mohri, and Ananda Theertha Suresh. FedBoost: A communication-efficient algorithm for federated learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the International Conference on Machine Learning (ICML)*, volume 119, pages 3973–3983. PMLR, 13–18 Jul 2020.

[102] Xin Yao, Tianchi Huang, Chenglei Wu, Rui-Xiao Zhang, and Lifeng Sun. Federated learning with additional mechanisms on clients to reduce communication costs. *CoRR*, abs/1908.05891, Aug 2019.

[103] Xiaofei Wang, Yiwen Han, Victor Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of edge computing and deep learning: A compre-

hensive survey. *IEEE Communications Surveys Tutorials*, 22(2):869–904, Jan 2020.

[104] Jed Mills, Jia Hu, and Geyong Min. Communication-Efficient federated learning for wireless edge intelligence in IoT. *IEEE Internet of Things Journal*, 7(7):5986–5994, Jul 2020.

[105] Sin Kit Lo, Qinghua Lu, Chen Wang, Hye-Young Paik, and Liming Zhu. A systematic literature review on federated machine learning: From a software engineering perspective. *ACM Computing Surveys*, 54(5), Jun 2021.

[106] Yunfan Ye, Shen Li, Fang Liu, Yonghao Tang, and Wanting Hu. Edgefed: Optimized federated learning based on edge computing. *IEEE Access*, 8:209191–209198, 2020.

[107] Anuran Mitra, Soumita Biswas, Tinku Adhikari, Arindam Ghosh, Soumalya De, and Raja Karmakar. Emergence of edge computing: An advancement over cloud and fog. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7, 2020.

[108] Mohammad Salehi and Ekram Hossain. Federated learning in unreliable and resource-constrained cellular wireless networks. *CoRR*, abs/2012.05137, 2020.

[109] Latif Khan, Walid Saad, Zhu Han, Ekram Hossain, and Choong Seon Hong. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Communications Surveys & Tutorials*, 23(3):1759–1799, 2021.

[110] Yiwei Li, Tsung-Hui Chang, and Chong Yung Chi. Secure federated averaging algorithm with differential privacy. In *Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, Oct 2020.

[111] Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Chenlin Huang, and Wei Song. FedProf: Optimizing federated learning with dynamic data profiling. *CoRR*, abs/2102.01733, Feb 2021.

[112] Sunghwan Park, Yeryoung Suh, and Jaewoo Lee. FedPSO: Federated learning using particle swarm optimization to reduce communication costs. *MDPI Sensors Journal*, 21(2), Jan 2021.

[113] Xin Yao, Tianchi Huang, Chenglei Wu, Rui-Xiao Zhang, and Lifeng Sun. Towards faster and better federated learning: A feature fusion approach. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 175–179, Aug 2019.

[114] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 440–446, Sep 2017.

[115] Chuan Zhang and Weihong Xu. Neural networks: Efficient implementations and applications. In *Proceedings of the IEEE International Conference on ASIC (ASICON)*, pages 1029–1032, Oct 2017.

[116] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cedric Renggli. The convergence of sparsified gradient methods. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, volume 31. Curran Associates, Inc., 2018.

[117] Alon Brutzkus and Amir Globerson. Why do larger models generalize better? A theoretical perspective via the XOR problem. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the International Conference on Machine Learning (ICML)*, volume 97, pages 822–830, 09–15 Jun 2019.

[118] Kai Packhäuser, Sebastian Gündel, Nicolas Münster, Christopher Syben, Vincent Christlein, and Andreas Maier. Deep learning-based patient re-identification is able to exploit the biometric nature of medical chest x-ray data. *Scientific Reports*, 12(1):14851, 2022.

[119] Linda Wang, Zhong Qiu Lin, and Alexander Wong. Covid-net: a tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. *Scientific Reports*, 10(1):19549, Nov 2020.

[120] Md. Rezaul Karim, Till Döhmen, Michael Cochez, Oya Beyan, Dietrich Rebholz-Schuhmann, and Stefan Decker. Deepcovidexplainer: Explainable covid-19 diagnosis from chest x-ray images. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1034–1037, 2020.

List of Publication

Book Chapter

1. **Kundjanasith Thonglek**, Norawit Urailertprasert, Patchara Pattiyathanee, Chantana Chantrapornchai, "Damaged Vehicle Parts Recognition using Capsule Neural Network", Research Innovations and Trends on Computer Vision and Recognition Systems.

2. **Kundjanasith Thonglek**, Kohei Ichikawa, Chatchawal Sangkeettrkarn, Apivadee Piyatumrong, "Auto-Scaling System Apache Spark Cluster using Model-Based Deep Reinforcement Learning", Heuristics for Optimization and Learning.

Journal Article

1. **Kundjanasith Thonglek**, Kohei Ichikawa and Keichi Takahashi and Chawanat Nakasan and Kazufumi Yuasa and Tadatoshi Babasaki and Hajimu Iida, "Toward Predictive Modeling of Solar Power Generation for Multiple Power Plants," in IEICE Transactions on Communications, Jul. 2023.

2. **Kundjanasith Thonglek**, Keichi Takahashi, Kohei Ichikawa, Chawanat Nakasan, Hidemoto Nakada, Ryousei Takano, Pattara Leelaprute, Hajimu Iida, "Automated Quantization and Retraining for Neural Network Models Without Labeled Data," in IEEE Access, vol. 10, pp. 73818-73834, Jul. 2022.

3. **Kundjanasith Thonglek**, Norawit Urailertprasert, Patchara Pattiyathanee, Chantana Chantrapornchai, "Vehicle Part Damage Analysis Platform for Auto insurance Application", ECTI Transactions on Computer and Information Technology (ECTI-CIT), vol. 15, no. 3, pp. 313–323, Nov. 2021.

Conference Paper

1. **Kundjanasith Thonglek**, Thanaporn Jinnovart, Arnan Maipradit, "Privacy-Preserving Machine Learning for Snoring Detection", IEEE International

Conference on Information and Education Technology, Mar. 2023.

2. **Kundjanasith Thonglek**, Keichi Takahashi, Kohei Ichikawa, Chawanat Nakasan, Pattara Leelaprute, Hajimu Iida, "Sparse Communication for Federated Learning", IEEE International Conference on Fog and Computing, May. 2022.

3. **Kundjanasith Thonglek**, Kohei Ichikawa, Keichi Takahashi, Kazufumi Yuasa, Tadatoshi Babasaki, Chawanat Nakasan, Hajimu Iida, "Enhancing the Prediction Accuracy of Solar Power Generation using a GAN", IEEE International Conference on Green Energy and Smart Systems, Nov. 2021.

4. Sopicha Stirapongsasuti, **Kundjanasith Thonglek**, Shinya Misaki, Yugo Nakamura, Keiichi Yasumoto, "INSHA: Intelligent Nudging System for Hand Hygiene Awareness", ACM International Conference on Intelligent Virtual Agents, Sep. 2021.

5. **Kundjanasith Thonglek**, Kohei Ichikawa, Kazufumi Yuasa, Tadatoshi Babasaki, "LSTM-based Neural Network Model for Predicting Solar Power Generation", Technical Committee on Energy Engineering in Electronics and Communications (IEICE-EE), May. 2021.

6. **Kundjanasith Thonglek**, Keichi Takahashi, Kohei Ichikawa, Chawanat Nakasan, Hajimu Iida, "Federated Learning of Neural Network Models with Heterogeneous Structures", IEEE International Conference on Machine Learning and Applications, Dec. 2020.

7. Thanaporn Jinnovart, Xiongcai Cai, **Kundjanasith Thonglek**, "Abnormal Gait Recognition in Real-Time using Recurrent Neural Networks", IEEE International Conference on Decision and Control, Dec. 2020.

8. Sopicha Stirapongsasuti, **Kundjanasith Thonglek**, Shinya Misaki, Bunyapon Usawalertkamol, Yugo Nakamura, Keiichi Yasumoto, "A Nudge-based Smart System for Hand Hygiene Promotion in Private Organizations", ACM International Conference on Embedded Networked Sensor Systems, Nov. 2020.

9. **Kundjanasith Thonglek**, Keichi Takahashi, Kohei Ichikawa, Chawanat Nakasan, Hidemoto Nakada, Ryousei Takano, Hajimu Iida, "Retraining Quantized Neural Network Models with Unlabeled Data", IEEE International Joint Conference on Neural Networks, Jul. 2020.

10. **Kundjanasith Thonglek**, Kohei Ichikawa, Keichi Takahashi, Chawanat Nakasan, Hajimu Iida, "Improving Resource Utilization in Data Centers using an LSTM-based Prediction Model", HPCMASPA in IEEE International Conference on Cluster Conference, Sep. 2019.

11. **Kundjanasith Thonglek**, Norawit Urailertprasert, Patchara Pattiyathanee, Chantana Chantrapornchai, "IVAA: Intelligent Vehicle Accident Analysis System", IEEE International Joint Conference on Computer Science and Software Engineering, Jul. 2019.

12. **Kundjanasith Thonglek**, Kohei Ichikawa, Chatchawal Sangkeettrkarn, Apivadee Piyatumrong, "Auto-Scaling Apache Spark Cluster using Deep Reinforcement Learning", International Conference on Optimization and Learning, Jan. 2019.

Poster Presentation

1. **Kundjanasith Thonglek**, Kohei Ichikawa, Keichi Takahashi, Hajimu Iida, "Federated Learning Infrastructure for Collaborative Machine Learning on Heterogenous Environments", PRAGMA Student Workshop, Dec. 2021.

2. **Kundjanasith Thonglek**, Kohei Ichikawa, Keichi Takahashi, Chawanat Nakasan, Hajimu Iida, "SharingNets: an open-source repository for sharing pre-trained neural networks", IST-FR, Dec. 2019.

3. **Kundjanasith Thonglek**, Kohei Ichikawa, Keichi Takahashi, Chawanat Nakasan, Hajimu Iida, "Compressing Recurrent Neural Network Models using Vector Quantization", PRAGMA Workshop 37, Sep. 2019.

4. Vahid Daneshmand, Renato Figueiredo, Kohei Ichikawa, Keichi Takahashi, **Kunjanasith Thonglek**, "Investigating the Performance and Scalability

of Kubernetes on Distributed Cluster of Resource-Constrained Edge Devices", PRAGMA 37, Sep. 2019.

5. **Kundjanasith Thonglek**, Kohei Ichikawa, Keichi Takahashi, Chawanat Nakasan, Hajimu Iida, "Towards Optimal Resource Utilization in Data Centers using Long Short-Term Memory", PRAGMA Workshop 36, Apr. 2019.