CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

ORCA – Online Research @ Cardiff

This is an Open Access document downloaded from ORCA, Cardiff University's institutional repository:https://orca.cardiff.ac.uk/id/eprint/157633/

This is the author's version of a work that was submitted to / accepted for publication.

Citation for final published version:

Amaizu, Maryleen Uluaku, Ali, Muhammad, Anjum, Ashiq, Liu, Lu, Liotta, Antonio and Rana, Omer 2023. Edge-enhanced QoS aware compression learning for sustainable data stream analytics. IEEE Transactions on Sustainable Computing 8 (3) , pp. 448-464. 10.1109/TSUSC.2023.3252039

Publishers page: http://dx.doi.org/10.1109/TSUSC.2023.3252039

Please note:
Changes made as a result of publishing processes such as copy-editing, formatting and page numbers may not be reflected in this version. For the definitive version of this publication, please refer to the published source. You are advised to consult the publisher's version if you wish to cite this paper.

This version is being made available in accordance with publisher policies. See http://orca.cf.ac.uk/policies.html for usage policies. Copyright and moral rights for publications made available in ORCA are retained by the copyright holders.

Edge-Enhanced QoS Aware Compression Learning for Sustainable Data Stream Analytics

M U Amaizu, M Ali, A Anjum, L Liu, A Liotta, O Rana

Abstract—Existing Cloud systems involve large volumes of data streams being sent to a centralised data centre for monitoring, storage and analytics. However, migrating all the data to the cloud is often not feasible due to cost, privacy, and performance concerns. However, Machine Learning (ML) algorithms typically require significant computational resources, hence cannot be directly deployed on resource-constrained edge devices for learning and analytics. Edge-enhanced compressive offloading becomes a sustainable solution that allows data to be compressed at the edge and offloaded to the cloud for further analysis, reducing bandwidth consumption and communication latency. The design and implementation of a learning method for discovering compression techniques that offer the best QoS for an application is described. The approach uses a novel modularisation approach that maps features to models and classifies them for a range of Quality of Service (QoS) features. An automated QoS-aware orchestrator has been designed to select the best autoencoder model in real-time for compressive offloading in edge-enhanced clouds based on changing QoS requirements. The orchestrator has been designed to have diagnostic capabilities to search appropriate parameters that give the best compression. A key novelty of this work is harnessing the capabilities of autoencoders for edge-enhanced compressive offloading based on portable encodings, latent space splitting and fine-tuning network weights. Considering how the combination of features lead to different QoS models, the system is capable of processing a large number of user requests in a given time. The proposed hyperparameter search strategy (over the neural architectural space) reduces the computational cost of search through the entire space by up to 89%. When deployed on an edge-enhanced cloud using an Azure IoT testbed, the approach saves up to 70% data transfer costs and takes 32% less time for job completion. It eliminates the additional computational cost of decompression, thereby reducing the processing cost by up to 30%.

Index Terms—Deep Autoencoders, Cloud Computing, Data compression, Edge Computing, Quality of Service, Real-time analytics, Transmission Optimisation

1 Introduction

The amount of data generated from IoT devices, wearables, and sensors is rapidly growing in scale and velocity for applications such as smart healthcare, smart cities, and video stream analytics. In centralised data center based learning and analytics, all incoming raw data are transmitted to the cloud (data center) for processing, leading to high communication overhead, latency and energy consumption. On-device inference, on the other hand, deploys compressed deep neural networks (DNNs) on mobile devices. However, *over compression* can cause serious performance degradation and limited resources can lead to high communication latency across devices.

Edge and fog computing [1] have emerged as a solution to pre-process data at the edge of the network, reducing the cost of transmission and latency [2], [3]. Machine learning algorithms have gained attention for pre-processing massive data streams en-route to cloud data centres in distributed networks [4], [5]. These tasks range from simple processing tasks like metadata extraction to more complex tasks like anomaly detection and object recognition [6]. One such pre-processing technique is compression of data at the edge of the network for transmission to the cloud, known as compressive offloading [7]. Edge-enhanced compressive offloading has become a sustainable solution that allows data to be compressed at the edge and offloaded to the cloud for further analytics, thus reducing bandwidth consumption and communication latency.

Compressive offloading techniques face the challenge of finding a sustainable model that meets changing Quality of Service (QoS) requirements such as performance, bandwidth and data quality. Real-time systems often need to adapt to changes in resource availability to meet their application-specific QoS requirements that vary significantly at run-time [8], [9], [10]. QoS re-negotiation is often needed to adapt to dynamic workload, where users can dynamically change QoS requirements depending on the level of QoS required [11].

An example of a use case for compressive offloading is a multimedia application in which a user specifies that the video quality should be low during a video conversation, but should be improved quickly when a new person joins the conversation [12]. Another example is a video conferencing application that only requires the best quality video/ audio for the person speaking.

Existing approaches for determining the best model for compressive offloading includes performing empirical analysis to select the best model for compression [13], [14]. The best model is selected by evaluating the performance of a cloud model on data reconstructed using different compres-

- *M. U. Amaizu, M. Ali, A. Ashiq, L. Liu, are with the School of Computing and Mathematical Sciences, University of Leicester, UK. Emails: mun1@leicester.ac.uk, ma909@leicester.ac.uk, a.anjum@leicester.ac.uk, l.liu@leicester.ac.uk*
- *O. Rana is with the School of Computer Science and Informatics, Cardiff University, UK. Email: ranaof@cardiff.ac.uk*
- *A. Liotta is with the Faculty of Computer Science, Free University of Bozen-Bolzano, Italy. Email: Antonio.Liotta@unibz.it*

Manuscript received Month Day, Year; revised Month Day, Year.

sion models. However, these models are fixed at run-time, so they cannot respond to dynamic QoS demands. Additionally, the empirical analysis is computationally expensive and does not follow a principled search strategy. Methods such as [15] focus on hardware constraints of the device and compress the deep learning model to meet QoS requirements, but do not compress the data. Another approach is to choose an inferior but efficient local model to eliminate or cut down data transmission [16]. This requires a tradeoff between accuracy and speed, leading to a significant drop in accuracy. Other data compression techniques, such as filtering and pre-processing, have been applied to produce metadata for offloading [17]. However, these techniques may not be useful in applications where data needs to be reconstructed at the server side or where a classifier needs to be trained on the original data or its meaningful representations.

This paper addresses these gaps by designing and implementing a compression learning method for discovering the compression models that offer the *right* QoS for an application. The approach maps features to models and classifies them for a range of QoS models. The search strategy is based on a narrowed search over the entire neural architectural space to reduce the computational cost of hyperparameter search. An automated QoS-aware orchestrator is deployed to select the best model for compressive offloading on the edge. It works as a tuning engine to achieve the desired compression at runtime. For example, if the machine learning task is simple, it can be tuned to the maximum level of compression to transmit the smallest size of data (Figure 1(a)). If the machine learning task is complex, the system compresses lightly to achieve a better compromise (Figure 1(b)). There will be many different combinations in between. The complexity of a task is subjective and application-specific and must be defined for the task. The orchestrator has the flexibility and adaptability for practitioners to tune the whole chain to achieve the required performance on a machine learning task. The system defines QoS constraints as a combination of different metrics, such as the quality of reconstructed data, resource availability at the edge, bandwidth availability and target accuracy.

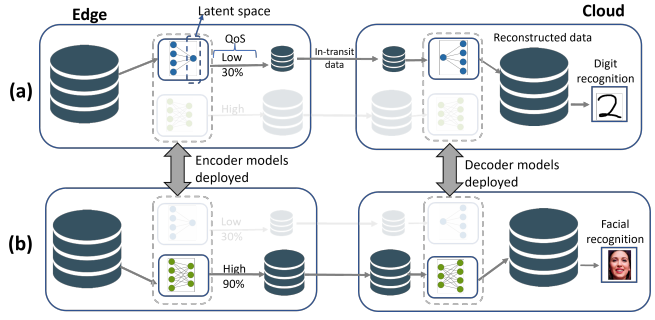


Fig. 1: QoS-aware data compression

Deep autoencoder [18] is a type of deep learning model we adapted for compression learning and compressive offloading based on its unique characteristics – such as latent space splitting point, portable encodings and fine-tuning capability. The edge-enhanced configurations we proposed are based on the modification of autoencoder to address the possible scenarios of edge-cloud collaboration, namely EdgeCompress-CloudDecompress (ECCD) and EdgeCompress-Only (ECO). The latter eliminates the additional computational complexity of decompressing data in the cloud and is useful where the main objective is to perform the machine learning task. Experiments are conducted to demonstrate how intelligent data compression can increase potential capacity on the cloud to carry out other workloads. We measure the computational complexity and overheads incurred by our approach for real-time analytics and show how they map to QoS requirements. Two kinds of data streams and motivating use cases were analyzed, namely object classification and human activity recognition. The main contributions of this work can be summarised as follows:

1) A principled compression learning method has been designed and implemented to learn models for different QoS requirements. It takes advantage of the redundancy in the features to reduce the search space (Section 3).
2) An automated QoS-aware orchestrator has been designed to select, in real-time, the best autoencoder model for compressive offloading in edge-enhanced clouds based on changing QoS requests of users. The orchestrator is designed to have diagnostic capabilities to search appropriate parameters that give the best compression. The system is capable of processing a large number of QoS requests in a given time due to the discoverable pool of QoS models and features. The implemented edge-enhanced cloud saves up to 70% data transfer cost, and takes 32% less time for job completion (Section 4).
3) A modularisation approach has been proposed and implemented to map features to models and classifies them for a range of QoS models. This finds the model that can achieve the lowest compression at the expected QoS without compromising user-defined QoS. The system removes up to 99% data redundancy without losing desirable performance.
4) Our proposed approach is novel in harnessing the capability of autoencoders for edge-enhanced compressive offloading based on latent space splitting, portable encodings and fine-tuning network weights. The approach has been validated on an Azure IoT test bed in which the EdgeCompress-Only configuration eliminates the additional computational cost of decompression, thereby reducing the processing cost by up to 30%.

2 Background and Preliminaries

In this section, we review literature related to model partitioning, feature encoding and model selection to provide context for the solution described in the paper. We discuss how our approach differs or improves upon existing solutions in the field of deep learning-based data compression and offloading. Specifically, we highlight how existing solutions lack the ability to effectively adapt models to changing QoS conditions in large-scale streaming systems.

2.1 Model partitioning with encoder-decoder models

Previous studies have examined the application of machine learning in partitioning and optimizing bandwidth

usage through the deployment of various encoder-decoder models. Yao et al. [19] utilized deep encoder-decoder models to perform compressive offloading on mobile devices, with the partitioning decision being based on latency estimates. Shao and Zhang [20] employed co-inference based on feature compression, with an encoder designed to be adaptive to different channel conditions. QoS considerations were made by training the encoder with different channel conditions to improve its generalization ability. However, while deep neural networks (DNNs) can gradually extract intermediate features layer by layer, the feature dimension may not decrease. This can lead to an "in-layer data amplification" phenomenon, where the output data size of early layers may be larger than the original input data [21]. Ko et al. [22] introduced an edge-host partitioning method that combines model partitioning with lossy feature encoding. This means that the intermediate data after model partitioning is compressed using lossy feature encoding before transmission. Simulation results showed that this approach leads to significant improvements in energy efficiency and throughput compared to configurations that perform the entire inference at the edge or at the host. The feature coding method applied in this study used JPEG and Huffman coding to compress the intermediate data, which may remove high-frequency components of the features that are important for the classification task. On the other hand, Sbai et al. [23] proposed a partitioning approach for DNN inference between the edge and the cloud, which they claim to be the first work to consider simultaneous optimization of both the memory usage at the edge and the size of the data to be transferred over the wireless link. However, this work proposes injecting a hand-crafted compression module to reduce feature communication, which can lead to high engineering and training costs, as well as sub-optimal system performance and accuracy degradation. In addition to partitioning methods based on encoder-decoder models, recent research also includes federated learning, such as [24] that proposed a method to divide the process into the model and exemplar stages, effectively addressing the catastrophic forgetting problem.

In contrast to the above-mentioned approaches, our model is dynamic and utilizes a QoS orchestrator. This orchestrator selects the best model after the edge node is updated on the QoS requirements. Additionally, a copy of the model is deployed on both the edge and cloud to avoid increased computational load during inference. This research focused on training models through hyperparameter optimization, with the latent space as a parameter, and selecting the best compression models for runtime based on accuracy and image quality.

2.2 Feature encoding

Existing literature has shown that feature encoding through dimensionality reduction on the edge or mobile device can help achieve the goals of real-time data stream analytics [25], [26] and data transmission optimization [27], [28], [29], [30]. However, many dimensionality reduction methods suffer from high information loss and data distortion, making it difficult to have a true representation of the data. Linear methods like Principal Component Analysis (PCA) are not able to capture the high non-linearity in large-scale datasets generated by IoT applications. Additionally, traditional data compression techniques like JPEG are beyond the scope of this work, as they belong to the class of image compression algorithms that use heuristic-based blocks of encoders and decoders.

Autoencoders [18], a special type of neural network, have gained widespread attention for their ability to learn meaningful representations in large-scale data streams [31]. They remove redundancy by constraining the bottleneck layer of the network (also known as the latent space dimension) and have become attractive for data compression through dimensionality reduction, as seen in studies such as [3], [14], [27], [30], [32], [33]. Autoencoders have been used to perform feature extraction and data compression in various applications, including image [34], [35] and video analytics [36]. Autoencoders are particularly useful as an alternative to linear models like PCA as they can map non-linear relationships and learn the structure of data and correlations between input features when forcing the input through the network's bottleneck.

Existing approaches, such as [13], incur heavy computational overheads by using an empirical approach to dimensionality reduction/ selection. In addition, these algorithms lack experimental analysis of the effect of data compression on real-time analytics and are fixed, not adapting to changing QoS conditions in real-time. In our work, we pose dimensionality selection as a model selection problem in order to tune the parameters of the autoencoder and select the best model for data compression within acceptable QoS constraints.

2.3 Model selection

In recent years, techniques for selecting the best model that meets system requirements from a set of possible models have been posed as a hyperparameter optimisation problem. Classical hyperparameter optimisation algorithms like grid search, random search and genetic/ evolutionary algorithms have been used to find optimal neural network architecture for convolutional neural networks [37]. These methods evaluate the cost of the defined metrics either by real-time execution of each architecture or using *approximators*. A comprehensive survey of hardware-aware neural architecture search is provided in [38], where the best model is selected based on hardware capabilities of the device or platform.

Yao et al. [25] defined a search space consisting of different partitions of deep encoder-decoder models for performing compressive offloading on mobile devices. They performed an offline training of these models on a deep learning engine while profiling and modelling the execution time with FastDeepIoT [19]. During runtime, the best-partitioned model is selected depending on the current system capabilities, measured by end-to-end latency and bandwidth. The goal was to reduce overhead on the resource-limited end device and to put most of the computation on the server side. The authors limited QoS to hardware constraints of the end device and applied an existing dynamic offloading strategy.

Lu et al. [15] presented QoE compression models for deep learning. Despite their focus on QoE, their work differs from ours in that they focus on deep learning model

compression rather than data compression. Therefore, while their approach addresses latency and performance issues, it does not address transfer optimization.

3 QoS-aware Compression Learning

The goal of QoS-aware compression learning is to find the optimal balance between compression and QoS requirements for a specific application. We use the example of an image recognition task to illustrate this concept. In this task, recognizing large objects in the image may not require high resolution, but for finer details such as facial features in face recognition, a higher resolution is necessary. If the machine learning task is simple, we can opt for a higher level of compression to transmit a smaller amount of data, resulting in lower quality reconstruction. However, if the task is more complex, such as in face recognition, we may choose to compress less to achieve a better balance between compression and quality. There are many variations in between these two extremes. The orchestrator provides the flexibility and adaptability for practitioners to adjust the entire process to meet the desired performance for the machine learning task.

It must be recognised that QoS metrics can have different implications and effects on data compression. The system should be designed to set priority levels and the right combinations of metrics to achieve the desired results. Multimedia transmission is typically affected both by the network performance and application QoS parameters. It is essential to monitor not only the network performance parameters such as packet delay and minimum bandwidth, but also the application parameters such as image quality and frame rate, which can ultimately affect the user's QoE. These are subjective and can vary depending on the application and users [39]. An example of QoS definitions for an image recognition use case is provided in Table 1. The QoS constraints can be defined with a combination or ranking of different metrics such as quality of reconstructed data, resource availability at the edge, bandwidth availability and target accuracy. For example, the system may be tasked with achieving a low QoS based on an accuracy of 70% which requires heavy compression of up to 90% of the data. However, if resources are not available at the edge or if the user specifies how much resources they can contribute, the system may bypass the compression task and send the full data or minimally compressed data to the cloud. Similarly, if the edge has sufficient resources for compression, but the channel bandwidth availability is high, the system can override the original compression rate and compress as much as the bandwidth will permit.

TABLE 1: Example of QoS definitions for an image recognition use case

QoS metric	Definition
Resource availability	How much memory, energy and processing are available at the edge
Bandwidth	How much data can go through the link at any given time
Accuracy	The performance of the reconstructed data on the cloud machine learning task
Scalability	Resizing data without losing image quality
Image quality	The quality of reconstructed image

3.1 Compression learning using autoencoders

In current autoencoder designs, a trade-off exists between compression size and compression loss in connection to data quality, which strongly depends on this bottleneck layer. This can be fine-tuned as desired, based on the architecture design of the network [40]. Autoencoders are good for compression learning and compressive offloading for the following reasons: (i) The latent space of autoencoders creates a natural splitting point for edge and cloud networks, (ii) Encodings carry most important information about the data at a reduced size which allow them to be portable, and (3) Trained autoencoders can be decoupled into stand-alone encoder and decoder networks.

3.1.1 Portable encodings

In order to design efficient autoencoders, our goal was to reduce the dimensionality of the encoded data while still maintaining the essential features of the input data as close as possible. A typical architecture for an autoencoder is illustrated in Figure 2. The process begins by feeding data, x, into the encoder. At the latent space, the data is compressed into a lower-dimensional representation. In the decoding phase, this compressed data is then reconstructed back to its original dimensions by the decoder. During training, the decoder is trained to learn the most important features that should be preserved in the lower-dimensional representation. The output of each layer is passed on to the next input layer before reaching the output layer. Among the hidden layers, the layer with the least number of neurons is referred to as the bottleneck. The first layer captures first-order features, the second layer captures second-order features, and so on. Stacked Autoencoders (SAE) are designed to capture hierarchical abstractions of knowledge and can effectively introduce compression by adhering to design specifications:

1) Input data must be compressed to a level that is appropriate for the network's capacity and identity.
2) The compression process should be adaptable to the efficiency requirements of the application.
3) The computational load should be distributed between edge nodes and the cloud.

By adjusting the number of neurons in the bottleneck layer, different levels of compression can be achieved. The number of neural layers in the network can also be optimized to improve the decoding process and to balance the technical and cost burdens between all nodes during training and decoding.

The deep compression is performed with a 2D input stream of dimension $r \times c$ transformed to a 1D stream of dimension $r * c$. Where $r, c \in \mathbb{N}$ are the number of rows and columns of the data stream respectively. For instance, a data sample of 28×28 pixels is transformed to 784 pixels during pre-processing. By applying the dimensionality constraint on the encoder network, we obtain $\mathbb{N}^{r*c} \xrightarrow{Enc} \mathbb{N}^{d}$, where d is the latent space dimension.

Equation 1 represents the compression rate (CR), which is defined as the data reduction in size relative to the uncompressed size of the data. It is the difference between

the input dimension and the dimension of the output of the encoder. The Compression ratio (CR) is given as:

$$CR = |\frac{d-D}{D}| \times 100\% \quad (1)$$

Where d is the dimensions of the input at the bottleneck layer and D is the size of the original data input dimensions.

Given training data up to k samples $\{x_1, x_2, ...x_k\}$, the autoencoder's objective is to reduce the reconstruction error (loss):

$$J(W,b) = \frac{1}{k}\sum_{i=1}^{k}(\frac{1}{2} \parallel x_i - \tilde{x_i} \parallel^2) \quad (2)$$

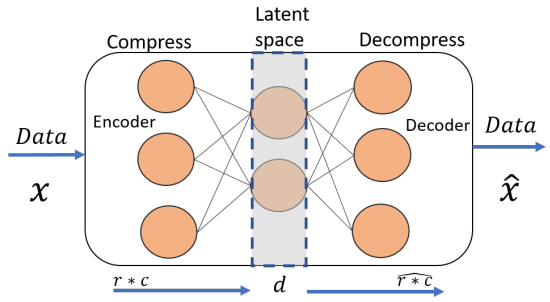


Fig. 2: Compression learning model architecture. The latent space is the point of splitting the model for offloading.

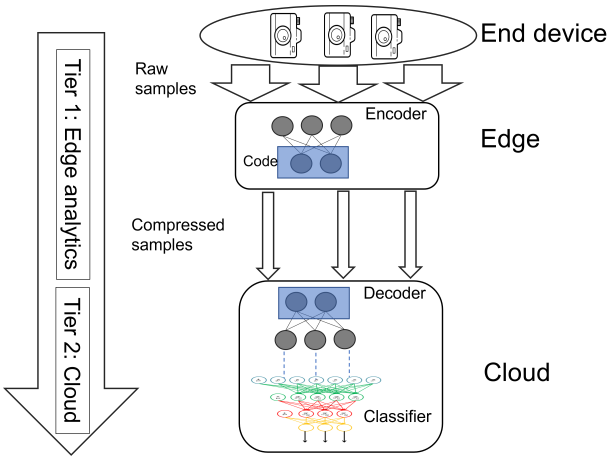


Fig. 3: Split autoencoder showing offloading of encoder compression model on the edge and decompression decoder on cloud

3.1.2 *Latent space splitting*

The latent space of autoencoders creates a natural splitting point for edge and cloud networks. In a split-autoencoder as shown in Figure 3, the encoder and decoder networks of the autoencoder are deployed on separate nodes of the distributed system. The intermediate data is compressed at the network edge by the encoder to reduce network traffic across the communication channel [27], while the decoder part may be deployed on the cloud to reconstruct the data.

3.1.3 *Fine-tuning of network weights*

Fine-tuning is a supervised learning task that adjusts the weights of the neural network to improve its classification performance. The learning process can be further customized so that the cloud server can accept data in either fully reconstructed or encoded form. The two methods for fine-tuning autoencoders for compression learning are discussed below.

EdgeCompress-Only (ECO): Decompression of the data can add additional computational overhead to the overall analytics. In ECO, both autoencoder and cloud machine learning models are trained together but the decoder is discarded afterward. The joint offline training is to ensure the cloud model architecture was suitable for the output of the encoder. The configuration is shown in Figure 4 (a). In terms of neural network architecture, the encoder is simultaneously connected to the decoder and the cloud classifier model. The mean squared loss and cross-entropy are simultaneously minimized between the encoder and decoder, and classifier respectively. This eliminates the need to decompress the data in the cloud during inference, as the cloud model is trained with the output of the encoder and can still make inferences directly with the encodings. This configuration is effective in scenarios where the outcome of the machine learning task is of primary importance, and the original form of the data is not necessary as long as it is represented in a meaningful way.

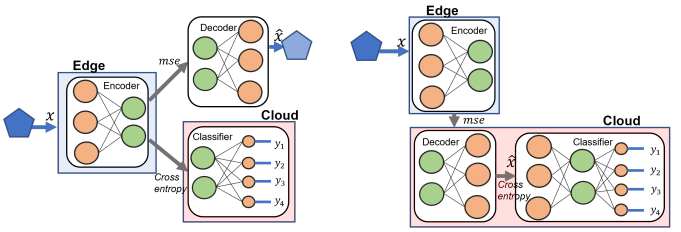
EdgeCompress-CloudDecompress (ECCD): In the ECCD configuration shown in Figure 4 (b), autoencoder and cloud model are integrated for the supervised training using the combined models' loss objective given in Equation 2. The encoder is connected to the decoder which is then connected to the cloud classifier model. During training, the mean squared loss and cross-entropy losses are sequentially minimized between encoder and decoder, and decoder and classifier respectively. During the inference phase, the encoder part of the autoencoder is deployed on the edge while the decoder part is coupled with the cloud. After decompressing the data back to the original shape, the data is passed to the cloud model for the classification task. In this configuration, the decoder is a necessary component on the cloud as it enables the reconstruction of the image into the original shape for which the cloud model has been trained. This configuration is favourable when data needs to be returned back to the original shape for monitoring at the cloud or when there may be a need to use another classifier trained with the original data.

3.2 Principled method for discovering model pool

Compression learning requires finding the optimal compression model architectures based on their performances on evaluation metrics such as accuracy. This can be posed as a hyperparameter optimisation problem and have been recently applied to neural architecture search in [37]. Here, we will provide details of the design process in terms of search space, search strategy and performance estimation.

3.2.1 *Search space*

The simple search space is the space of chain-structured neural network. The architecture for DNN model A can be written as a sequence of n layers, where the i'th layer L_i



(a) EdgeCompress-Only (ECO) (b) EdgeCompress-CloudDecompress (ECCD)

Fig. 4: Supervised compression learning combines autoencoder and classifier. The decoder is discarded in ECO after learning. Models are deployed after training on the edge or cloud or discarded in the case of ECO decoder

receives its input from layer $i-1$ and its output serves as the input for layer $i+1$, i.e., $A = L_n \cdot ...L_1 \cdot L_0$. The search space is parametrised by: n number of layers; type of every operations performed by layer (e.g., convolutional, pooling, fully-connected layers); hyperparameters of every layer tied to the operation (e.g., strides, kernel size, filters).

For this one-dimensional model, we have limited our search space to the number of neurons in the bottleneck layer as this is the main component that affects data quality and performance after compression. The latent space dimension, d, which is the number of neurons in the bottleneck layer of autoencoder, is chosen as a hyperparameter to optimise. It begins by defining a search space of possible latent space dimensions from 1 to maximum dimensions of the dataset, d_n, in increments of defined step size. This is designed for deep learning models that require a single numerical value for latent space, for example, the stacked autoencoder.

3.2.2 Search strategy

The search strategy advances from the traditional method of hyperparameters optimisation, which makes a complete search over a given subset of the hyperparameters space of the training algorithm. The pseudocode of the search strategy is presented in Algorithm 1. It works in a similar fashion as the conventional binary search except that the middle element is the latent space and QoS is the target and we can shift right or left depending on whether the target value is lower or higher. Binary search seeks a single value so it uses the middle element as a pivot to move in the right or left direction; here, the middle element is the latent space dimension which is always stored and continues to move in the best direction to keep adding QoS-features pair to the pool. The target middle QoS (Q_m) is computed by running the model training at the middle dimension (d_m). The algorithm checks whether the middle QoS (Q_m) is equal to the search target QoS (Q_h) in every iteration. If Q_h is greater than Q_m, then the left half or elements before the middle elements of the list is eliminated from the search space, and the search continues in the remaining right half. Else if Q_h is less than Q_m, the right half elements or all the elements after the middle element is eliminated from the search space, and the search continues in the left half. This process is repeated until the upper dimension becomes less than the lower dimension We use a one-dimensional search space that includes only the number of neurons in the bottleneck layer for simplicity. Though our work specifies the search space to be the dimensions of the bottleneck layer, other hyperparameters could still be added to the search space. For example, for 2D and 3D convolutional autoencoders, the bottleneck layer consists of multiple hyperparameters (kernel size, strides, filters) rather than only the number of hidden neurons in the 1D stacked autoencoders.

3.2.3 Performance estimation

The objective of the search strategy discussed above is to find a neural architecture a that maximises some performance measure. As the search space is relatively small, this keeps the computational cost manageable as the number of architectures to be explored are limited. Hence, performance estimation was done by performing a standard training and validation of the architecture on data. Based on historical conditions for the application, the QoS can be estimated for a given model selection decision on an edge device. The QoS parameters can be influenced by multiple metrics.

In our experimentation, accuracy and data quality were two metrics that could affect QoS. Although other factors, such as the DNN model architecture and latency can also be taken into account, we simplify things by assuming that the DNN model accuracy follows a linear function, $accuracy = f_1(K_{node}, K_{layer})$. Additionally, the reconstruction error of the model has an impact on data quality which can affect the user experience in certain application. The data quality is denoted as $dataquality = f_2(K_{node}, K_{layer}, E_d)$ for unsupervised learning, where E_d is the distance between two points on the plane. Thus, when K_{node}, K_{layer}, E_d, CPU, and RAM are taken into account as a whole, the expected QoS can be modeled as a linear function with an unknown parameter θ that can be learned online. The context vector $x_{t,A}$ captures all the available side information, including selected features of both the DNN model A and the incoming edge device such as the DNN model size, architecture, device's CPU, and RAM capacity. Let Q be the QoS of the DNN model A selected at time period t. Since Q is a random variable relying on context $x_{t,A}$, we model it as:

$$\mathbb{E}[Q, A] = f(x_{t,A}) = x_{t,A}^T \theta \tag{3}$$

For simplicity, We assume that the expected QoS is a linear function of data quality and accuracy for unsupervised and supervised learning respectively, even though users' preferences for these metrics can vary greatly.

The above shows how the QoS metrics can be defined and how the metrics are quantifiable to a single number for Algorithm 2 to select the appropriate model. QoS is subjectively defined given that QoS can vary for different domains/users/datasets. Based on the application requirements, resource availability, and network conditions, more complicated functions may be used to quantify the QoS. Once the architectural search space is selected, training does not depend on specific hardware, software, or network and is done once for each application except where significant concept drift occurs.

Algorithm 1 QoS-aware Compression learning

Input: input x, epoch e, Total dimensions $\{D \in \mathbb{N} | D > 0\}$
Output: QoS-features map, P
Initialisation:
  d_l = 1 ▷ Lowest dimension
  d_h = D ▷ Highest dimension
  $\theta\{A\}, Q_h \leftarrow ModelTraining(D, e)$
modularise (Q, d):
  if Q does not exists **then**
    Create new Q subset
  end if
  Append d to Q subset
SearchLatentSpace (d_l, d_h):
  $k \leftarrow d_i$ ▷ Set intrinsic dimension
  if $d_h \geq d_l$ **then**
    $d_m = (d_h + d_l)/2$▷ Mid dimension
    $\theta\{A\}, Q_m \leftarrow ModelTraining(d_m, e)$
    modularise (Q_m, d_m)
    if $Q_m < Q_h$ **then**
      SearchLatent (d_m, d_h)
    else
      SearchLatent (d_1, d_m)
    end if
  end if
  return P

3.3 Modularisation of compression models

The compression models are modularised by mapping and grouping them to features (dimensions) depending on the QoS, Q. This way, the model that achieves the highest or lowest compression from that subset can be selected. Equation 4 describes the entire QoS-features super set, P. Where $Q_{\{1,i\}}$ refers to the QoS-features set for lowest QoS, Q_1, for the group of features, d ranging from i_1 to i_n. For autoencoder models, these features represent the number of latent space dimensions in the bottleneck layer at that QoS. Through modularisation, the system can adapt to different QoS demands by selecting the features that meet the demand per QoS request.

$$P = \begin{array}{c|ccc} Q & 1 & \cdots & n \\ \hline Q_{\{1,i\}} & \langle \mathrm{d}_{q_1,i_1} \rangle & \cdots & \langle \mathrm{d}_{q_1,i_n} \rangle \\ \cdots & \cdots & \cdots & \cdots \\ Q_{n-1,j} & \langle \mathrm{d}_{q_{n-1},j_1} \rangle & \cdots & \langle \mathrm{d}_{q_{n-1},j_n} \rangle \\ Q_{n,k} & \langle \mathrm{d}_{q_n,k_1} \rangle & \cdots & \langle \mathrm{d}_{q_n,k_n} \rangle \end{array} \quad (4)$$

4 Automated QoS-aware Orchestrator

The work of the orchestrator is to select and deploy trained autoencoder models for QoS-aware data compression at the edge in real-time. Figure 5 shows how the orchestrator is set up for the QoS-aware dynamic model selection for data compression. During deployment, only one model is active at a time to compress the data. The edge node is updated whenever a new QoS requirement is identified and the orchestrator selects the model that matches the QoS specification. However, any edge-based QoS model specification like resource availability will be determined at the edge node. To select the right cloud decompression model, the metadata specifying compression level and the in-transit data will be sent to the cloud. The solution is based on the assumption that the QoS metrics can be predicted as the streams arrive. Where the incoming streams have different resolutions, they are rescaled to the input dimension of the dataset. The autoencoder layer takes the same input dimension because it has been designed for that dataset. Intelligent compression can be performed at edge nodes, saving bandwidth, reducing data processing time, and, in a few cases, improving accuracy. However, the model must be trained at sufficient granularity to perform optimized compression. Model training and optimized model search is an overhead. We assume that there is enough redundancy in the dataset and that not all features are important; the goal is to extract only important features to speed up the inference task without losing accuracy. An edge node can be any resource from a Raspberry Pi, PC, laptop, or virtual machine. A cloud node can be any resource on the Internet that can be accessed remotely.

4.1 Compression model selection and deployment by Orchestrator

To deal with the variable QoS conditions, orchestrator is designed to select the best possible compression model from the QoS-features map P to be deployed on the nodes. Assuming there are a set of D possible latent spaces, we select the model with features/dimensions d that meets the required QoS in real-time. During inference, the orchestrator selects the best model based on online QoS conditions as per Algorithm 2. If a new QoS request is received during a batch streaming, then the new requirements are fed into the equation which calculates a new value for QoS. From the QoS-features map P, the minimum dimension is fetched for the corresponding QoS value. Orchestrator then communicates the chosen dimension d_m to the nodes which will then take the trained stored model that corresponds to d_m from their database and deploy on the node.

To handle changing QoS conditions, the orchestrator is designed to select the best possible compression model from the QoS feature map P to deploy at the nodes. Given a set of possible latent spaces D, it selects a model with features/dimension d that satisfies the online QoS conditions using the Algorithm 2. If a new QoS request is received during batch streaming, a QoS value is estimated using the available information fed into Equation 3. Then the lowest feature/dimension that gives the estimated QoS value is selected from the QoS feature map P. The selected dimension d_m is sent to the nodes (edge, cloud), which then retrieves the stored trained model corresponding to d_m and deploys it on the node.

The system can be applied to different datasets, but must first be trained to find the best latent dimensions. Once the system is trained, it should not need to repeat the process of finding the best latent dimensions again. It is also worthy of note that the best dimensions of two datasets can differ, hence, models must be trained on specific datasets pertaining to the application. As with any real-time system with offline model training, a mechanism is required to

Algorithm 2 Automated QoS-aware orchestrator

Input: input streams S, QoS-features map P
Output: Compression model selected
for each $bufferedstream$ in S **do**
  if nth QoS request is received: **then**
    Fetch new requirements, r:
    Compute Q_n using Equation 3
    {Retrieve the corresponding dimension, d_m from the QoS-features map}
    $d_m = min(P : Q_n \rightarrow d)$
    Select trained model pairs that corresponds to d_m
    Deploy model pairs (encoder and decoder) on associated nodes
    Deploy selected model pairs for d_m.
    Compress $bufferedstream$ with new model A_i
  else
    Compress $bufferedstream$ with existing model A_{i-1}
  end if
end for

monitor for concept drift. The reconstruction error (Equation 2) between original data and reconstructed data allows monitoring for significant concept drift. An application-specific threshold can be set for the application to trigger retraining whenever the new data samples significantly drift from the distribution.

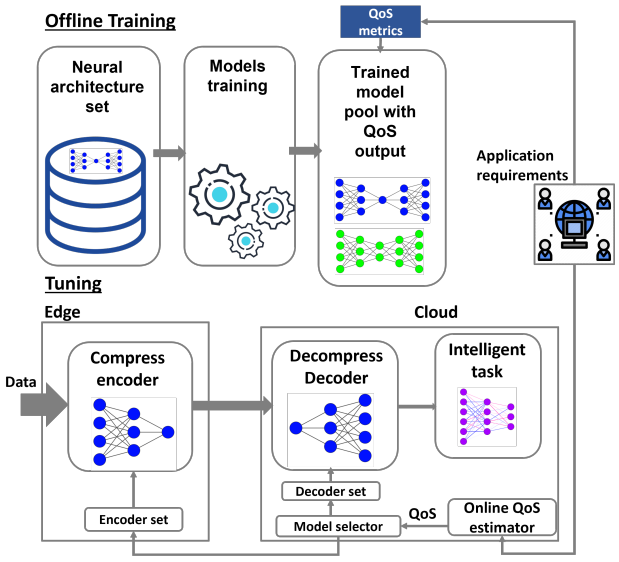


Fig. 5: Functional illustration of the compression learning phase (offline training) and the compressive-offloading phase (tuning) performed by the automated QoS-aware orchestrator. The tuning is for selecting the best compression model that meets QoS requirements in real-time

4.2 Diagnostic Capabilities of the QoS-aware Orchestrator

The orchestrator can perform diagnostics by querying the QoS-features map P to extract insights of the application domain and corresponding performance of the system. This capability allows it to search appropriate parameters that give the best compression. It provides more understanding about how the compression should be done. A number of desired information which the orchestrator can obtain for diagnostic and prescriptive purposes are highlighted.

1) Desired performance: From the QoS-features map, the orchestrator can determine how many features must be added to attain a desired performance. Moving from a performance of $Q_{\{1,i\}}$ to $Q_{n-1,j}$ could mean increasing the number of features from $\langle \mathrm{x}_{q_1,i_1} \rangle$ to $\langle \mathrm{x}_{q_{n-1},j_1} \rangle$.
2) Compression range: It addresses the question: *how far can data compression stretch at the same QoS*? By adding additional features to the dataset, the accuracy can be improved. From feature set P, the compression range for i group of features can be represented as:

$$|\langle \mathrm{x}_{q_1,i_1} \rangle - \langle \mathrm{x}_{q_1,i_n} \rangle|_{Q_{\{1,i\}}} \tag{5}$$

3) QoS range: The QoS range for the given dataset and application can be determined from P. The QoS range will help to inform the users on how best to tune the model and how good the expected results would be. The QoS range for P QoS-features map is given as:

$$Q_{n,k} - Q_{\{1,i\}} \tag{6}$$

4) Domain understanding: Based on the QoS range we can gain insights on whether the data is coming from the same distribution, in which case it will advise how much of compression is necessary. For example, a high QoS range means that compression will not be very useful as the data may be coming from highly different distributions.

4.3 Cost of edge-enhanced compressive offloading

The services for the proposed edge-enhanced data compression can be decomposed into three levels (a) compress, (b) decompress, (c) classify. The time cost to process one data stream in the edge-enhanced model, C_p is given as:

$$C_p = E_t + D_t + C_t + T_t \tag{7}$$

Where E_t is the encoder's time cost for compressing data, D_t is the decoder's time cost for decompressing the encoded data (only applies for the ECCD configuration), C_t is the classifier's time cost to classify the frames into labels, and T_t is the time to transfer data from the data source to the cloud. The data transmission cost approximates to:

$$C_T = \sum_{n=1}^{N} (T_{c_n} + C_{p_n}) \tag{8}$$

Where T_{c_n} the total data transfer time for transferring N data samples from edge to cloud and C_{p_n} is the time cost of processing N data samples.

The data transmission efficiency of the architecture a is defined in terms of percentage gain with respect to data transmission cost of the cloud architecture , c.

$$E_a = \frac{C_T(c) - C_T(a)}{C_T(c)} \times 100 \tag{9}$$

Where $C_T(a)$ is the total time cost to transmit all the streams in architecture a and $C_T(c)$ is the total time cost to transmit all the streams on the traditional cloud architecture.

5 IMPLEMENTATION

The proposed implementation posits the utilization of the Azure IoT platform, in conjunction with a serverless architecture and B-series VMs. It must be noted, however, that this approach may not be well-suited for other types of platforms or resources. Furthermore, the use of B-series VMs may potentially impede network performance and may not be appropriate for workloads that necessitate consistent, full performance of the CPU. The concept of an edge node can encompass a wide range of devices, such as a Raspberry Pi, personal computer, laptop, or even a VM in the cloud. However, the generalizability of the proposed QoS compression learning method is limited, as it has only been defined in terms of accuracy and data quality, and has only been applied to two datasets in the domains of object recognition and human action recognition. Additionally, the QoS is contingent upon various factors, such as latency and bandwidth, among other system requirements. Therefore, these intelligent compression learning techniques must be tailored to user-defined application parameters. The classifier model employed in the experimental use cases is a basic 5-layer convolutional neural network. Another assumption of this approach is that there is a sufficient degree of redundancy in the dataset, and that not all features are essential; the aim is to extract only the most important features without sacrificing accuracy in order to expedite the inference task.

5.1 System configurations

We set up three configurations for experimentation namely Cloud-Only (CO), EdgeCompress-Only (ECO), and EdgeCompress-CloudDecompress (ECCD). Both ECO and ECCD configurations are referred to as edge-enhanced configurations. These configurations have been explained from the learning aspects in Section 3.1 as the output of the coupling of autoencoder and cloud classifier during the compression learning. In this section, the details of the implementation will be discussed. The configurations demonstrate the effects of edge-enhanced analytics on application-level QoS measures like bandwidth and latency. As shown in Figure 6 the analytics process is decomposed into 3 subtasks namely Compression (T1), Decompression (T2), and Classification (T3).

5.1.1 The Cloud-Only (CO) Configuration

In this configuration, all the raw image data is sent to the cloud. The raw data is directly transmitted from the end devices to the cloud for the classification task (T3). There is no intermediate processing or edge node involved in the analytics. This represents the conventional analytics approach where no intermediate processing is done on the edge nodes. Then, the machine learning classifier is applied to raw data as shown in Figure 6 (Cloud). The result of the classification is the label of the images (L). We compute the execution time, bandwidth cost, and accuracy of ML classification.

5.1.2 The EdgeCompress-Only (ECO) Configuration

In this configuration, the raw data is first sent to the edge where the data is compressed (T1) as shown in Figure 6 (ECO). The data is then sent to the cloud for classification (T3). There is no decompression (T2). The resulting labels are produced from the classification task (L). All the raw data will queue up to be compressed at the edge and only the compressed data is sent to the cloud. During training, the encoder is connected to the decoder as well as to the classifier. The mean squared loss between encoder and decoder and the cross-entropy loss between the encoder and classifier are jointly minimized. The decoder is discarded afterward. This configuration works in a setting where the end result from the machine learning task is paramount not the actual data, thus, data is not needed to be in its original form but can be in some form of meaningful representations

5.1.3 The EdgeCompress-CloudDecompress (ECCD) Configuration

In this configuration, the raw data is first sent to the edge where the data is compressed (T1) as shown in Figure 6 (ECCD). The data is then sent to the cloud for decompression (T2) and machine learning classification task (T3). The resulting labels are produced from the classification task (L).

During training, the encoder is connected to the decoder, while the decoder is connected to the classifier. The mean squared loss between encoder and decoder, and cross-entropy loss between decoder and classifier are jointly minimised. The decoder is a necessary component of the cloud as it enables the reconstruction of the image back to its original dimensions. This configuration is favourable when data needs to be returned back to the original shape for monitoring at the cloud or when there may be a need to use another classifier trained with the original data.

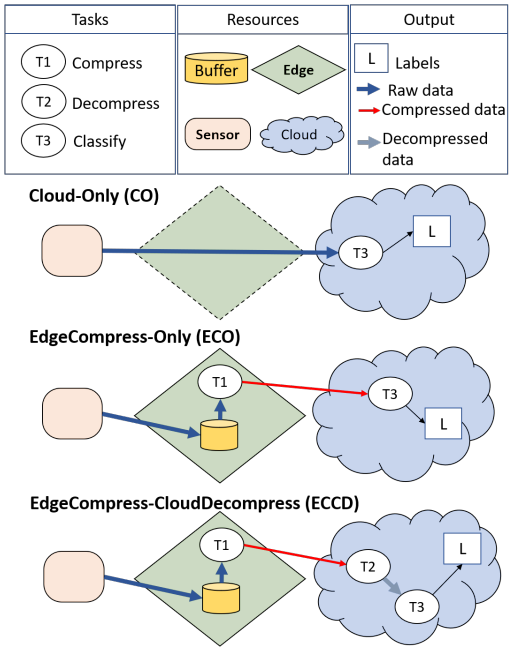


Fig. 6: Three system configurations. CO is the baseline for cloud-centric analytics. ECO and ECCD are edge-enhanced configurations differing by how the autoencoder components are coupled during compression learning and compressive offloading

5.2 Experimental testbed

Experiment was conducted on Azure IoT platform using virtual machines (VMs) and serverless architecture to deal

with servers and computing resource management. Azure IoT is a public cloud platform providing the benefit of handling big data and access from anywhere in the world. We run a set of experiments for the baseline and edge-enhanced compression configurations. The Cloud-Only and edge-enhanced configurations are shown in Figure 7 and Figure 8 respectively, with both using the serverless architecture to manage data processing. The edge module comprises a streaming application that sources data and sends data streams to the cloud storage. For the edge-enhanced configurations, the data stream is first sent to the compression module where the intelligent compression algorithm compresses the data and transfers the compressed data to the cloud storage. The above configurations are run using a Virtual Machine (VM) located in Western Europe (UK) as the edge device. The Edge VM is deployed in the Western Europe region to ensure enough distance from the remote data centre. The Edge VM memory is set to 2GB with 1 virtual CPU.

On the cloud side, the Azure cloud services in the East US region (Virginia) are chosen. An Azure Cloud VM that uses geo-redundant storage option RA-GRS for blob storage in the cloud is provisioned. This Azure storage serves as the cloud database which receives the incoming data stream and serves as a data ingestion source for the cloud processing engine. The cloud VM is created as a resource in this same region with the cloud database. The Cloud VM memory is set to 8GB with 2 virtual CPUs.

The B-series VMs used for both edge and cloud can be deployed on a variety of hardware types and processors, so competitive bandwidth allocation is provided. The B series is variable, and its design does not provide the user with a consistent level of network performance. B-series VMs are ideal for workloads that do not need the full performance of the CPU continuously, like web servers, proof of concepts, small databases, and development build environments. They run on the 3rd Generation Intel® Xeon® Platinum 8370C (Ice Lake), the Intel® Xeon® Platinum 8272CL (Cascade Lake), the Intel® Xeon® 8171M 2.1 GHz (Skylake), the Intel® Xeon® E5-2673 v4 2.3 GHz (Broadwell), or the Intel® Xeon® E5-2673 v3 2.4 GHz (Haswell) processors. B-series are economical virtual machines that provide a low-cost option for workloads that typically run at a low-to-moderate baseline CPU utilization, but sometimes need to burst to significantly higher CPU utilization when the demand rises. B1ms VM's threshold is 20% CPU Utilization, so every hour you are under 20% you are gaining credits, every hour you are over you are "bursting" and consuming credits. If you configure a VM to use 2 vCPUs with 2 cores when you have a physical processor whose clock speed is 3.0 GHz, then the total clock speed is 2x2x3=12 GHz.

The cloud application is deployed as a containerised flask app to provide API service to a serverless function. The flask application performs classification on the images ingested into the cloud database. In the ECCD configuration, this module also contains the decompression algorithm for decompressing images before the classification task. The different configurations are given their unique addresses so that they can be uniquely called by their respective serverless functions. The flask application is containerised for easier provisioning and deployment on the virtual machine.

The Azure Functions (AF) service is provisioned as a serverless function that checks when a new data sample is ingested to the database and makes a GET request to the Cloud VM for decompression and/or classification. The serverless function gets triggered whenever a new image is ingested into the cloud database. We set the number of images to be uploaded to a total of 100 image files to ensure the system is not congested. When a huge volume of images is uploaded quickly, many functions are activated at once. This causes the output to be out of sequence and affects the computation of statistics.

The approach used on Azure IoT platform can be adapted to other cloud platforms such as Amazon Web Services (AWS) and Google Cloud Platform (GCP) by using similar technologies like virtual machines, serverless functions, and containerisation. For example, AWS offers EC2 instances and Lambda functions, while GCP has Compute Engine VMs and Cloud Functions. These platforms offer similar features for computing resource management and data processing, which can be leveraged for similar configurations as described for Azure IoT. However, the specific virtual machine configurations and cloud services used may differ depending on the platform.

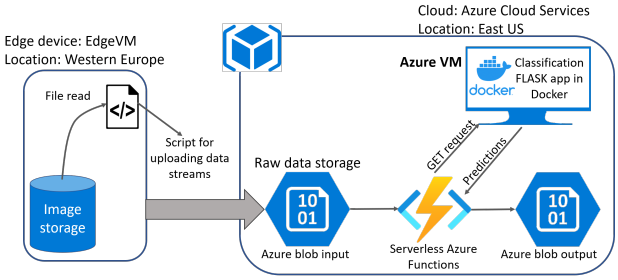


Fig. 7: Schematic of a Cloud-Only configuration on Azure environment

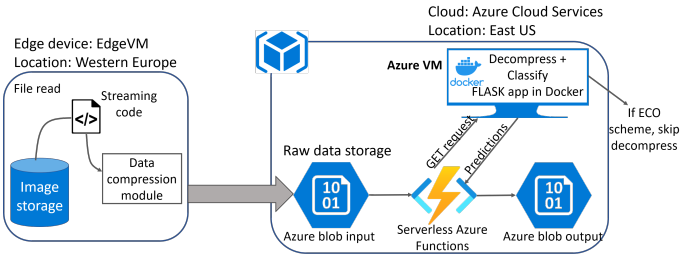


Fig. 8: Schematic of an edge-enhanced configuration on Azure environment

5.3 Model Training and Deployment

In the experimentation, the models trained include the Cloud-Only (classifier trained on raw data), EdgeCompress-Only (ECO), EdgeCompress-CloudDecompress (ECCD), Autoencoder (AE), and Principal Component Analysis (PCA). Both ECO and ECCD models are trained together with the classifier to achieve a supervised fine-tuning. AE is an autoencoder model trained in an unsupervised manner independent of the classifier and then tested with the Cloud-Only classifier model. It is a way to check how well the

autoencoder can reconstruct the data and perform well on the classification task without using any supervised objective. The parameters for training the autoencoder and classifier models as well as their architectures are given in Table 2 for reproducibility. A cloud classifier designed with 1D convolutional layers takes in a dataset reshaped into the form of *Number of samples* $\times$ *total dimension of the data sample* to produce one of the output values.

During deployment, for the ECO configuration, the encoder model is deployed on the edge to compress the data, while the decoder model is discarded after training. For the ECCD, the decoder model is attached to the cloud classifier and both are deployed on the cloud. For the Cloud-Only configuration, the classifier is deployed on the cloud and there are no encoder and decoder models.

TABLE 2: Parameters for machine learning model training

	Autoencoder	Classifier
Model architecture	StackedAE: 784-1000-500-250-d-250-500-1000-784 Where d is the latent dimension	Reshape(28,28,1)-Conv2D(32,3,3)-Conv2D(64,3,3)-MaxPooling2D(2,2)-Dropout(0.25)-Flatten-Dense-Dropout(0.5)-Dense
Loss function	MSE	Categorical cross entropy
Optimiser	Adam	SGD
Batch size	256	128
Training samples	60,000 (Fashion-MNIST), 7352 (HAR)	
Validation set	20% of Training set	
Datasets	Fashion-MNIST, HAR	
Training time	100 epochs	100

5.4 Datasets and use cases

5.4.1 Object classification (Fashion-MNIST)

There are numerous applications of object classification in computer vision tasks, such as semantic segmentation, scene understanding, image retrieval, fine-grained classification. We focus on multimedia applications where images need to be sent to far-away data centres for classification. The Fashion-MNIST dataset [41] consists of 60,000 images for the training set and 10,000 images for the test set. Every image has a row by column shape of 28x28 which amounts to 784 pixels. The dataset has similar dimensions as the more popular MNIST, and is designed for more challenging classification problems than MNIST. The intrinsic dimensionality is known to be the number of classes of both datasets which is 10. The aim is to recognise and classify images sent to the cloud into one of the 10 labels. Our approach required a dataset from which we can evaluate other search strategies that use the intrinsic dimensionality as an incremental step size. We also required a dataset that can be transformed into one-dimensional features to be fed into the stacked 1D autoencoder. Due to the fine-tuning of network weights, the dataset must also have classes that can be used for supervised learning. The experiments required testing our hypothesis that the compression model can be modularised into a QoS-features model pool with the assumption that the dataset must have significant redundancy. Based on these requirements, the FASHION dataset was picked for experimentation as they have been shown to possess adequate redundancy and have an intrinsic dimensionality of 10 which is equal to the number of classes. Furthermore, we consider different data types that are usually streamed in different applications like the HAR.

5.4.2 Human Action recognition (HAR)

Action recognition involves using sensors to read characteristics of the environment and the user in order to identify the activities. This is an interesting use case for several applications such as healthcare where persons could be remotely monitored to recognise their activities and provide personalised care. Another appealing use case is the assisted daily living (ADL) for the elderly or disabled. In computer vision tasks, video understanding, content searching and multimedia event detection are some of the interesting applications. We have selected the UCI HAR dataset [42] which was created from the readings of accelerometer sensors attached to the body of the participants and used to record the activities. The activities are categorised into one of these 6 set of physical activities: standing, walking, laying, walking, walking upstairs and walking downstairs.

5.5 Evaluation

We compare the proposed edge-enhanced compression learning against the existing baseline systems. The comparison is done on the basis of model types and search strategy.

1) **Model Types**: From literature, intelligent data compression is achieved broadly using linear or non-linear models. A conventional implementation of the edge computing paradigm is to train a linear model that compresses image and sends to the cloud server, where all the tasks are then executed. We consider a baseline referring to this compression scenario based on [43], [44]. Non-linear models rely on the non-linearity of high-dimensional and unstructured data to perform compression. We implemented a similar experiment that uses autoencoders [13]. We evaluate the model's performance in terms of the prediction performance of the cloud machine learning task.
2) **Search strategy**: The **empirical** search approach for model selection rely on trial and error which we implement as a search over all possible latent space dimensions. We also implement the **step** strategy based on compressing the model by regular percentile of compression rate.

6 Results and Discussion

This section reports the performance results of the proposed method in an edge-enhanced distributed system. The system configurations are tested against the system requirements: QoS awareness, scalability, latency, and transmission optimisation. The baseline algorithm is the cloud classifier trained on the raw data without an autoencoder. The Fashion-MNIST and HAR datasets are used for validation for application in object classification and action recognition respectively.

6.1 QoS-Aware Compression Learning

The first thing this experiment seeks to demonstrate is the redundancy in the data that necessitates modularisation

of the QoS-features pool. In this paper, we have used reconstruction quality and accuracy as the QoS estimator for unsupervised learning and supervised learning respectively.

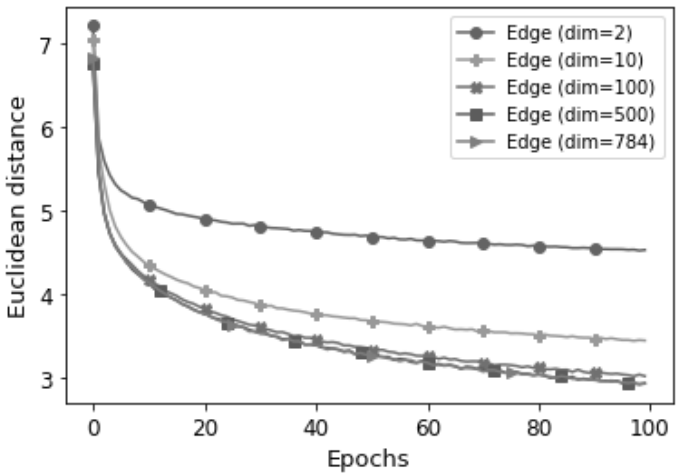


Fig. 9: Changing the dimensions is equivalent to changing the compression models. We see significant redundancy between 100, 500, 784 dimensions necessitating modularisation

6.1.1 QoS as reconstruction quality

For the unsupervised approach, the Euclidean distance was used as a QoS measure of the reconstructed image quality. A lower Euclidean distance approximates to a higher data quality as the difference between the original and the reconstructed data is low. Figure 9 shows how the Euclidean distance varies for latent space dimensions, d = 2, 10, 100 and 784. The Euclidean distance decreases as the dimensions increase. There is a larger difference in the Euclidean distance between d = 2 and d = 10 as compared to between $d = 100$ and $d = 784$. Also, the difference in Euclidean distance between $d = 2$ and $d = 100$ is very high compared to $d = 100$ and $d = 784$, despite the latter having a wider dimension range (648). This indicates that the effect of compression on the data quality becomes relatively low after a certain minimum dimension is reached. This minimum dimension is seen to capture most of the important features of the dataset. To estimate this dimension, $d = 784$ is used as a baseline for unsupervised algorithm. This dimension is expected to be the lowest distance to be realised with the architecture given that it has the same value as the input dimension. This set of autoencoder models mapping different latent space dimensions are trained by the orchestrator to map the different Euclidean distances as a QoS measure of reconstruction quality. Changing the dimensions is equivalent to changing the models. We see significant redundancy between 100, 500, 784. Hence, modularisation is relevant and changing QoS requests of users (reconstruction quality) can be met by switching the models.

6.1.2 QoS as accuracy

For supervised learning the measure of QoS used was accuracy. We compare the linear model Principal Component Analysis (PCA) against the non-linear model in

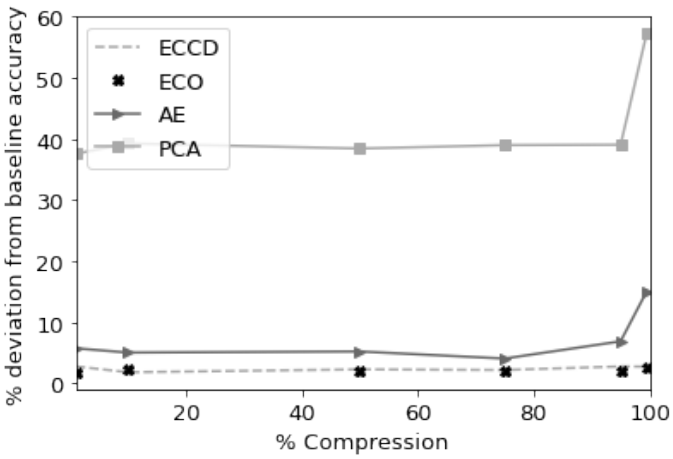


Fig. 10: HAR dataset compression learning: Delta accuracy at different latent space compression ratios shows significant redundancy eliminated by compression learning

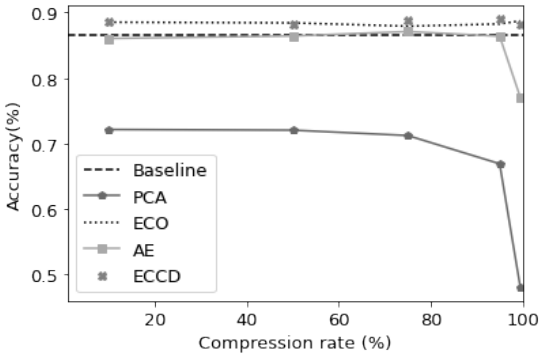


Fig. 11: Fashion-MNIST dataset compression learning: Models can be modularised and mapped to QoS achieved during training

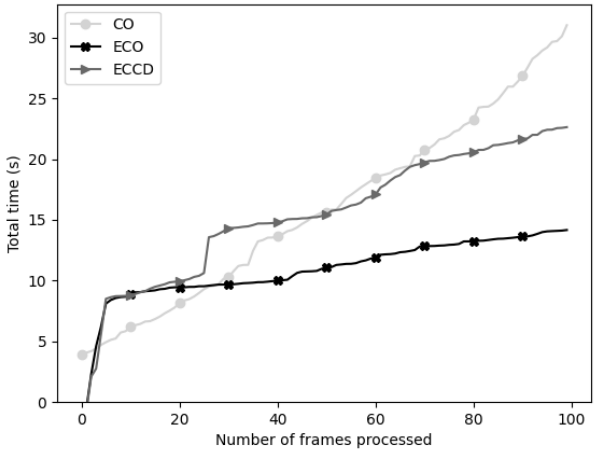


Fig. 12: Time taken to process frames is lowest at ECO

different configurations namely the baseline or Cloud-Only (classifier trained with raw data), EdgeCompress-Only (ECO), EdgeCompress-CloudDecompress (ECCD), Autoencoder (AE). Fine-tuning was done for ECO and ECCD to improve compression in accordance with the supervised objective. Certain compression percentage are sampled for this experiment. We chose the compression rates of 10, 50, 75, 95, 99.5 percentages. The compression ratios represent the different models trained by QoS-aware compression learning. The compression rate was calculated using Equation 1. The baseline remains fixed as there is no compression or decompression task performed in all the cases.

In Figure 10, we show the deviation from the baseline accuracy for different models trained on the HAR dataset. Here, 1% compression represents the minimum compression rate while 100% compression represents maximum compression rate. We noticed that for most of the compression rates the accuracy remains stable. This points to the fact that there is significant redundancy in the features between these two dimensions. With compression close to 100%, there is a significant reduction in accuracy for the two datasets, specifically between 95% and 99.5%. This indicates a high level of redundancy in about 99% of the data.

In Figure 11, the accuracy for different models trained with the orchestrator on the Fashion-MNIST dataset is shown. The supervised training approach also has the potential to be close to or to exceed the baseline accuracy. Both ECO and ECCD exceed the baseline accuracy in the Fashion-MNIST dataset, while in the HAR dataset they are very close to the baseline. For ECO where only the encodings are sent to cloud for inference, the accuracy is mostly the same as ECCD in both datasets. We can infer that learned encodings of the raw data perform equally well without the need for further decompression at the cloud. We investigate the ability of the system to resize data without losing data quality. For the ECCD configuration, the downscaling of the data into lower dimensional space and subsequent upscaling into original shape after data arrives at the cloud does not significantly affect the accuracy. The accuracy remains the same with increasing ratio of compression. In large-scale analytics, data may need to be resized as it moves from one medium to the other. The results show that the proposed configuration is scalable to accommodate such scenarios in streaming data transmission. Furthermore, scalability is measured by the change in latency for each data stream processed on the distributed pipeline. Figure 12 shows how speed and scale balance each other. We observe that the ECO configuration will allow for more jobs to be processed within the overall system time.

We can conclude from these experiments that with the QoS compression learning, the encoder-decoder model pairs can be mapped to corresponding test accuracies (QoS) achieved during training. During deployment, the system may require compression at a particular performance and the most suitable models can be selected.

6.1.3 Search Strategy

The search strategy is compared to other (traditional) approaches to search through the latent space dimensions. Table 3 compares the computational cost of different search strategies with our proposed approach. The baseline is a search space over all the latent spaces, from 1 to 784 for the Fashion-MNIST dataset. Apart from storing the models at runtime (which the other approaches did not perform) our search strategy reduces the computational cost by factoring in redundancy in the data during the search.

TABLE 3: Computational cost of Compression Learning (Fashion-MNIST dataset)

Reference implementation	Strategy	Total models	Time cost
Baseline	All latent space (empirical)	784	7563
[14], [45]	10% step	10	965
Ours	Binary search	9	868

6.2 QoS-aware Compressive Offloading

This section presents the practical implications of implementing Compressive Offloading with variable QoS requests (6.2.1) and with fixed QoS requests on a real test-bed (6.2.2).

6.2.1 Dynamic QoS requests

The computational cost of the orchestrator's decision to select the best model in real-time is measured in this experiment. The QoS request is randomly varied to show the computational cost of decision. Figure 13 shows the cost of a decision made by the orchestrator to select a specific model at runtime from the QoS-features map, P in Equation 4. From observation, changes to QoS requests are not expected to have much impact on latency as this only means a change to the compression model that was selected in real-time. At 1000 random QoS requests, the decision time of the orchestrator is linear with an increasing number of data streams. This shows that the orchestrator is not only able to handle a large number of varying QoS requests but also does this in the worst cases at a linear time.

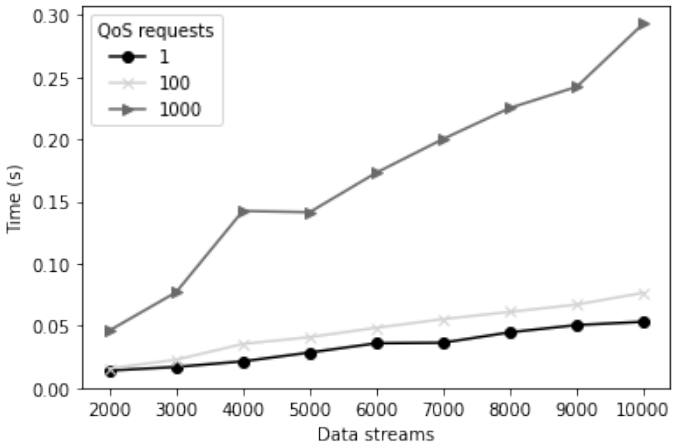


Fig. 13: The time it takes for QoS orchestrator to make decision on which model to deploy based on incoming QoS requests

TABLE 4: Data savings: Total input, payload and actual data transmitted in for the three different configurations

Config.	Total input size (MB)	Total raw payload size (MB)	Average transmitted dimension	% Data savings
CO	155.40	125.23	784	0%
ECO		39.93	39	69%
ECCD		39.99	39	69%

6.2.2 *Practical implications of compressive offloading on system-level QoS requirements*

Having established the cost of dynamic QoS requests, the QoS request is fixed to one in these sets of experiments to demonstrate the practical implications of deploying this system on a real-test bed. We show that centralised cloud-centric systems will have limitations in meeting system-level QoS, and thus are not efficient for compressive offloading in general.

1) **Bandwidth requirements**
In this experiment, we measure how much savings is achieved in transmitting data from edge to cloud for the three configurations. The same input data was transmitted in all the configurations. For the edge-enhanced configurations, the experiment was run using the reduced latent dimensions obtained during the algorithm training for the Fashion-MNIST dataset.

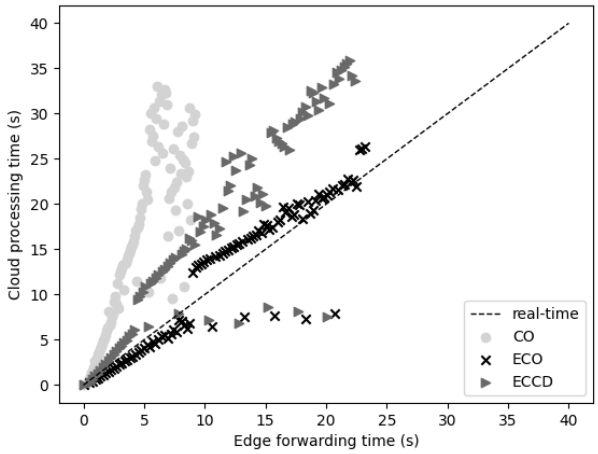


Fig. 14: Time to process single frame in all configurations. Individual data samples are represented by the markers.

Additionally, the data savings for the three configurations are computed in Table 4. The total input remains the same for the three configurations. For 100 frames, this amounts to 155 MB. The raw payload for ECO and ECCD is about 40 MB while that of CO is 4 times larger at 125 MB. It can be seen that the payload is highest for the cloud only configuration as all the streaming data produced per unit time is transferred to the cloud first and then analytics take place. Computing the percentage gain from Equation 9, the ECO and ECCD configuration are 69% more efficient in terms of transmission savings than the Cloud-Only configuration.

2) **Low-latency requirements**
This experiment will focus on showing the real-time implications of edge-enhanced analytics to satisfy the low-latency requirements of system QoS. In this experiment, we measure the time taken to complete jobs for all three configurations, Cloud-Only (CO), EdgeCompress-Only (ECO), EdgeCompress-CloudDecompress (ECCD). The real-time measurement is important to understand how much time is saved by compressing data at the edge and sending this across the network. It can be used to determine how much capacity can be created on the cloud for real-time analytics. Figure 14 plots the time taken for individual data streams to be transferred by the edge as against the time taken to be processed at the cloud. As expected, the Cloud-Only configuration has the shortest edge transmission time as no processing happened at the edge. However, the ECO configuration matches more closely to the real-time requirements of the system. Even though the data upload is done sequentially, the job completion is not sequential. Mostly in the Cloud-Only case, some of the data uploaded early on are processed at a later time. This indicates that the large size of the data stream may lead to additional delays in data arrival to the cloud. This has the potential to lead to a loss of the freshness of data which could limit real-time analytics and actuation decisions.
The total time cost for the compression, decompression, and processing is also computed in Figure 15 when the total number of frames equals 10, 50, and 100. We observe a huge growing cost of processing for the Cloud-Only configuration as the number of frames increase. We observe that the cost of ECCD started out higher than CO but later is lower than CO after the 100 frames are processed. From this we can infer that the size of the dataset has a significant impact on the cost of the cloud inference task. In this way, the ECO has a better advantage because the encodings are directly fed to the cloud classifier without incurring additional overhead of decompression.
Given all the results above, the ECO is the most efficient configuration for the QoS-aware edge-enhanced offloading with the lowest latency and least time to complete all jobs. For real-time or near real-time applications, the time to decompress the encodings must be minimal in order to pass the data to the cloud machine learning model and return the predictions to the user within a short space of time. Using percentage gain from Equation 9, this ECO configuration is 32% more efficient than the CO configuration

7 Conclusion and Future Directions

In centralised systems, large numbers of data streams can be sent to the data centres (clouds) for monitoring, storage and analytics leading to network congestion, high computational costs and latency concerns. Current approaches to compressive offloading involves deploying a fixed compression model which does not adapt to changing QoS. We proposed an automated QoS-aware orchestrator that selects and deploys trained deep learning models for data compression at the edge in real time. The orchestrator (Section 4) has been initialised by compression learning of a pool of autoencoder models. The models are defined

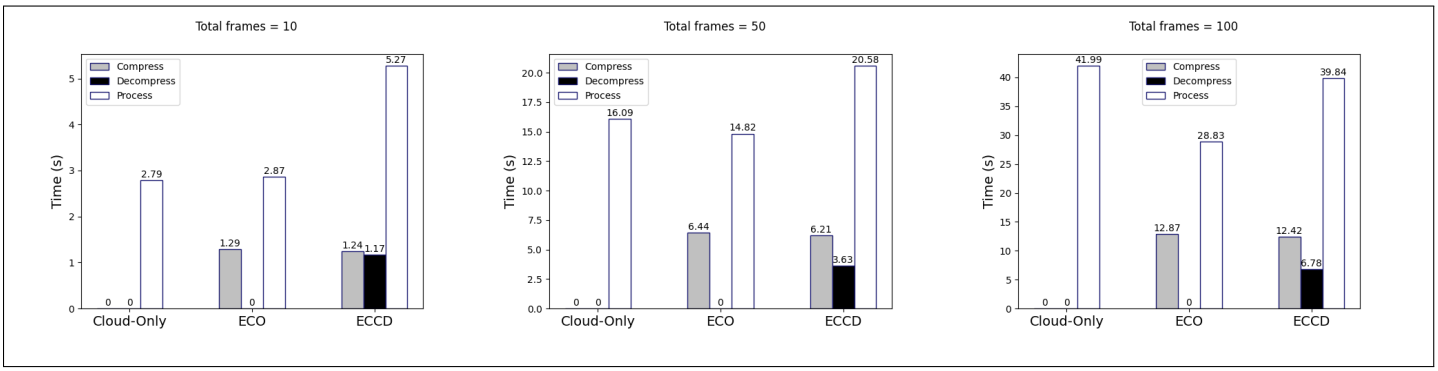


Fig. 15: Total time cost for compression, decompression, and processing for total frames = 10, 50 and 100

within a neural architecture search space and mapped to QoS in a modularised manner. The search space has been parametrised by the latent space dimensions of a stacked autoencoder. The search strategy involves a narrowed search over the neural architectural space, thereby reducing the computational cost of searching through the entire space by up to 89% while meeting the desired performance goals. It takes advantage of the redundancy in features to reduce the feature space.

The edge-enhanced configurations we proposed for compressive offloading are based on modifications of autoencoders to address the possible scenarios of edge-cloud collaboration, namely EdgeCompress-CloudDecompress and EdgeCompress-Only. The latter eliminates the additional computational cost of decompression and reduces processing cost by up to 30%. An experimental platform was set up on Microsoft Azure to demonstrate how the edge-enhanced configurations perform for the system requirements such as accuracy, scalability, latency and bandwidth as compared to the baseline centralised system configuration. The proposed system achieves up to 99% reduction in data size within acceptable performance. In addition, the data transfer cost was saved by up to 70% for the same amount of raw data and the system took 32% shorter time to complete the analytics. Although model learning can be performed offline in some environments, it can be prohibitive in distributed IoT systems due to data transfer costs, privacy and low latency requirements. In the future, we aim to enhance the capacity of large-scale streaming systems to support near real-time analytics via distributed learning. The edge-enhanced distributed learning system will support fully unsupervised systems that perform analytical tasks in real-time such as anomaly detection and clustering through analysing the difference between local and global models. Rather than sending data across the network, the data stays on the edge for privacy and real-time actuation. We believe that such systems will enable many practical real-time applications in IoT streaming systems where low-latency analytics are needed and where useful labels are rare or extremely difficult to collect. Future work will involve exploring dynamic tuning of autoencoders in the cloud and edge environments. Although we simplify the problem by limiting hyperparameter optimisation to the latent space dimension, other architectural hyperparameters can also be tuned (e.g., convolution degree and channel number), which affect compression ratio and overall performance.

REFERENCES

[1] M. Ali, A. Anjum, M. Usman Yaseen, A. Reza Zamani, D. Balouek-Thomert, O. Rana, M. Parashar, M. U. Yaseen, A. R. Zamani, D. Balouek-Thomert, O. Rana, and M. Parashar, "Edge Enhanced Deep Learning System for Large-Scale Video Stream Analytics," in *2018 IEEE 2nd International Conference on Fog and Edge Computing, ICFEC 2018 - In conjunction with 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE/ACM CCGrid 2018*, 2018, pp. 1–10. [Online]. Available: https://orca-mwe.cf.ac.uk/111447/1/Edge-Enhanced-Deep-Learning-ICFEC18.pdf

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] A. Abeshu and N. Chilamkurti, "Deep learning: The frontier for distributed attack detection in fog-to-things computing," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 169–175, 2018.

[4] H. Li, K. Ota, and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 1 2018.

[5] B. Tang, Z. Chen, G. Hefferman, T. Wei, S. Pei, Q. Yang, T. Wei, H. He, and Q. Yang, "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Transactions on Industrial informatics*, vol. 13, no. 5, pp. 2140–2150, 2017. [Online]. Available: https://www.researchgate.net/publication/313758226

[6] K. Guo, C. Shen, B. Hu, M. Hu, and X. Kui, "RSNet: Relation Separation Network for Few-shot Similar Class Recognition," *IEEE Transactions on Multimedia*, 2022.

[7] M. U. Ndubuaku, M. K. Ali, A. Anjum, A. Liotta, and S. Reiff-Marganiec, "Edge-enhanced analytics via latent space dimensionality reduction," in *2020 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*. IEEE, 2020, pp. 87–95.

[8] J. P. Loyall, A. K. Atlas, C. D. Gill, and D. L. Levine, "Flexible and Adaptive Control of Real-Time Distributed Object Computing Middleware," in *International Journal of Time-Critical Computing Systems*, 1999.

[9] G. Ismayilov and H. R. Topcuoglu, "Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing," *Future Generation computer systems*, vol. 102, pp. 307–322, 2020.

[10] R. E. Schantz, J. P. Loyall, C. Rodrigues, and D. C. Schmidt, "Controlling quality-of-service in distributed real-time and embedded systems via adaptive middleware," *Software: Practice and Experience*, vol. 36, no. 11-12, pp. 1189–1208, 2006.

[11] A. Abdelaal and H. Ali, "A New QoS Renegotiation Mechanism for Multimedia Applications," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. IEEE, 2005, pp. 305a–305a. [Online]. Available: http://ieeexplore.ieee.org/document/1385868/

[12] S. Weinstein, M. Suzuki, J. P. Redlich, and S. Rao, "A distributed object architecture for QoS-sensitive networking," in *1998 IEEE Open Architectures and Network Programming*. IEEE, 1998, pp. 3–13.

[13] A. M. Ghosh and K. Grolinger, "Deep Learning: Edge-Cloud Data Analytics for IoT," in *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*. IEEE, 2019, pp. 1–7.

[14] A. Z. Al-Marridi, A. Mohamed, and A. Erbad, "Convolutional Autoencoder Approach for EEG Compression and Reconstruction in m-Health Systems," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, 2018, pp. 370–375.

[15] B. Lu, J. Yang, L. Y. Chen, and S. Ren, "Automating deep neural network model selection for edge inference," in *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*. IEEE, 2019, pp. 184–193.

[16] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, IEEE. IEEE, 4 2018, pp. 1421–1429. [Online]. Available: https://ieeexplore.ieee.org/document/8485905/

[17] M. Ali, A. Anjum, O. Rana, A. R. Zamani, D. Balouek-Thomert, and M. Parashar, "RES: Real-Time Video Stream Analytics Using Edge Enhanced Clouds," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 792–804, 4 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9084281/

[18] P. Baldi, "Autoencoders, Unsupervised Learning, and Deep Architectures," in *JMLR: Workshop and Conference Proceedings 27:37–50, 2012 Workshop on Unsupervised and Transfer Learning*, 2012, pp. 37–50. [Online]. Available: http://proceedings.mlr.press/v27/baldi12a.html

[19] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 278–291.

[20] J. Shao and J. Zhang, "BottleNet++: An end-to-end approach for feature compression in device-edge co-inference systems," *2020 IEEE International Conference on Communications Workshops, ICC Workshops 2020 - Proceedings*, 6 2020.

[21] M. Chen, D. Gunduz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. Vincent Poor, "Distributed Learning in Wireless Networks: Recent Progress and Future Challenges," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 12 2021.

[22] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-Host Partitioning of Deep Neural Networks with Feature Space Encoding for Resource-Constrained Internet-of-Things Platforms," *Proceedings of AVSS 2018 - 2018 15th IEEE International Conference on Advanced Video and Signal-Based Surveillance*, 2 2019.

[23] M. Sbai, M. R. U. Saputra, N. Trigoni, and A. Markham, "Cut, Distil and Encode (CDE): Split Cloud-Edge Deep Inference," *Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks workshops*, vol. 2021-July, 7 2021.

[24] K. Guo, T. Chen, S. Ren, N. Li, M. Hu, and J. Kang, "Federated Learning Empowered Real-Time Medical Data Processing Method for Smart Healthcare," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2022.

[25] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM, 11 2020, pp. 476–488. [Online]. Available: https://dl.acm.org/doi/10.1145/3384419.3430898

[26] A. Kaur, P. Singh, and A. Nayyar, "Fog Computing: Building a Road to IoT with Fog Analytics," in *Fog Data Analytics for IoT Applications*. Springer, 2020, pp. 59–78.

[27] T. Mitani, H. Fukuoka, Y. Hiraga, T. Nakada, and Y. Nakashima, "Compression and aggregation for optimizing information transmission in distributed CNN," in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*. IEEE, 2017, pp. 112–118.

[28] A. Alahmed, A. Alrasheedi, M. Alharbi, N. Alrebdi, M. Aleasa, and T. Moulahi, "Machine Learning for Real-time Data Reduction in Cloud of Things," in *2020 2nd International Conference on Computer and Information Sciences (ICCIS)*. IEEE, 2020, pp. 1–6.

[29] C. Anagnostopoulos, "Edge-centric inferential modeling & analytics," *Journal of Network and Computer Applications*, vol. 164, p. 102696, 2020.

[30] E. Dasan and I. Panneerselvam, "A novel dimensionality reduction approach for ECG signal via convolutional denoising autoencoder with LSTM," *Biomedical Signal Processing and Control*, vol. 63, p. 102225, 2021.

[31] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *arXiv preprint arXiv:2003.05991*, 3 2020. [Online]. Available: http://arxiv.org/abs/2003.05991

[32] D. Wang, J. Ren, C. Xu, J. Liu, Z. Wang, Y. Zhang, and X. Shen, "PrivStream: Enabling Privacy-Preserving Inferences on IoT Data Stream at the Edge," in *2019 IEEE 21st International Conference on HPCC/SmartCity/DSS*. IEEE, 2019, pp. 1290–1297.

[33] T. Yu, X. Wang, and A. Shami, "UAV-Enabled Spatial Data Sampling in Large-Scale IoT Systems Using Denoising Autoencoder Neural Network," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1856–1865, 4 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8496746/

[34] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," *arXiv preprint arXiv:1703.00395*, 2017.

[35] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Deep convolutional autoencoder-based lossy image compression," in *2018 Picture Coding Symposium (PCS)*. IEEE, 2018, pp. 253–257.

[36] J. Pessoa, H. Aidos, P. Tomás, and M. A. T. Figueiredo, "End-to-end learning of video compression using spatio-temporal autoencoders," in *2020 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2020, pp. 1–6.

[37] P. Liashchynskyi and P. Liashchynskyi, "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS," 12 2019. [Online]. Available: http://arxiv.org/abs/1912.06059

[38] H. Benmeziane, K. E. Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "A Comprehensive Survey on Hardware-Aware Neural Architecture Search," *arXiv preprint arXiv:2101.09336*, 2021.

[39] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam, "Objective video quality assessment methods: A classification, review, and performance comparison," *IEEE transactions on broadcasting*, vol. 57, no. 2, pp. 165–182, 2011.

[40] S. Z. Valtchev and J. Wu, "Convolutional Autoencoders for Lossy Light Field Compression," *arXiv preprint arXiv:2008.00027*, 2020.

[41] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[42] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." in *Esann*, vol. 3, 2013, p. 3.

[43] A. Burrello, A. Marchioni, D. Brunelli, S. Benatti, M. Mangia, and L. Benini, "Embedded Streaming Principal Components Analysis for Network Load Reduction in Structural Health Monitoring," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4433–4447, 3 2021.

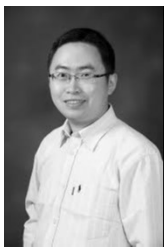
[44] T. Zhu, X. Cheng, W. Cheng, Z. Tian, and Y. Li, "Principal component analysis based data collection for sustainable internet of things enabled Cyber–Physical Systems," *Microprocessors and Microsystems*, vol. 88, p. 104032, 2 2022.

[45] A. Ben Said, A. Mohamed, T. Elfouly, K. Harras, and Z. J. Wang, "Multimodal deep learning approach for Joint EEG-EMG Data compression and classification," *IEEE Wireless Communications and Networking Conference, WCNC*, 5 2017.

Maryleen U. Amaizu earned a Bachelor's degree in Electrical and Electronic Engineering and a Masters degree in Embedded Systems. She also holds a Ph.D. from the University of Leicester, U.K. Her research is centered around enhancing distributed IoT systems for real-time IoT data processing, with a focus on using machine learning and edge computing. Her research interests also include deep learning, internet of things, and cloud/edge computing.

Muhammad K. Ali is a PhD student at the University of Leicester, UK. He has experience of working in diverse tools, technologies and programming languages. He has also worked in the software industry for about 5 years. His research interests include Edge and Fog computing, big data analytics, deep learning, intelligent multi-agents, high-performance computing and video analytics.

Ashiq Anjum is a professor of distributed systems at the University of Leicester, UK. Previously he was a professor of distributed systems and director of the data science research centre at the University of Derby, UK. Prof Anjum's areas of research include Data Intensive Distributed Systems, Distributed and Edge Enhanced Machine Learning models and Graph Learning Systems for High Performance Analytics.Recently he has been investigating edge enhanced self learning digital twins with applications in cyber-physical systems that have stringent quality of service requirements.



Lu Liu is a Professor and Head of the School of Computing and Mathematical Sciences at the University of Leicester, UK. Prof. Liu received his Ph.D. degree from the University of Surrey and M.Sc. degree from Brunel University. Prof. Liu's research interests are in the areas of data analytics, sustainable computing, service computing, artificial intelligence and the Internet of Things. He is a Fellow of British Computer Society (BCS).

Antonio Liotta is Full Professor at the Faculty of Computer Science, Free University of Bolzano (Italy), where he teaches Data Science and Machine Learning. Previously, he was the founding director of the Data Science Research Centre at the University of Derby, UK. Antonio´s research is focused on artificial intelligence theories and applications, particularly artificial vision, e-health, intelligent networks and intelligent systems. He is credited with over 350 publications involving, overall, more than 150 co-authors. Antonio is Editor-in-Chief of the Springer Internet of Things book series (springer.com/series/11636), and associate editor of several prestigious journals.

Omer Rana received the Ph.D. degree in neural computing and parallel architectures from Imperial College of Science, Technology and Medicine. He is a Professor of performance engineering at Cardiff University, U.K. His research interests include problem solving environments for computational science and commercial computing, data analysis and management for large scale computing, and high performance distributed computing (including edge and cloud computing).