Edge Intelligence : Empowering Intelligence to the Edge of Network

Xu, Dianlei

2021-11

Xu , D , Li , T , Li , Y , Su , X , Tarkoma , S , Jiang , T , Crowcroft , J & Hui , P 2021 , ' Edge Intelligence : Empowering Intelligence to the Edge of Network ' , Proceedings of the IEEE , vol. 109 , no. 11 , pp. 1778-1837 . https://doi.org/10.1109/JPROC.2021.3119950

http://hdl.handle.net/10138/341458
https://doi.org/10.1109/JPROC.2021.3119950

unspecified
acceptedVersion

Edge Intelligence: Empowering Intelligence to the Edge of Network

Dianlei Xu, Tong Li, Yong Li, *Senior Member, IEEE*, Xiang Su, *Member, IEEE*, Sasu Tarkoma, *Senior Member, IEEE*, Tao Jiang, *Fellow, IEEE*, Jon Crowcroft, *Fellow, IEEE*, and Pan Hui, *Fellow, IEEE*

Abstract—**Edge intelligence refers to a set of connected systems and devices for data collection, caching, processing, and analysis proximity to where data is captured based on artificial intelligence. Edge intelligence aims at enhancing data processing and protect the privacy and security of the data and users. Although recently emerged, spanning the period from 2011 to now, this field of research has shown explosive growth over the past five years. In this paper, we present a thorough and comprehensive survey on the literature surrounding edge intelligence. We first identify four fundamental components of edge intelligence, i.e. edge caching, edge training, edge inference, and edge offloading based on theoretical and practical results pertaining to proposed and deployed systems. We then aim for a systematic classification of the state of the solutions by examining research results and observations for each of the four components and present a taxonomy that includes practical problems, adopted techniques, and application goals. For each category, we elaborate, compare and analyse the literature from the perspectives of adopted techniques, objectives, performance, advantages and drawbacks, etc. This article provides a comprehensive survey to edge intelligence and its application areas. In addition, we summarise the development of the emerging research fields and the current state-of-the-art and discuss the important open issues and possible theoretical and technical directions.**

Index Terms—**Artificial intelligence, edge computing, edge caching, model training, inference, offloading**

I. Introduction

With the breakthrough of Artificial Intelligence (AI), we are witnessing a booming increase in AI-based applications and services. AI technology, e.g., machine learning (ML) and deep learning (DL), achieves state-of-the-art performance in various fields, ranging from facial recognition [1], [2], natural language processing (NLP) [3], [4], computer vision (CV) [5], [6], traffic prediction [7]–[9], and anomaly detection [10], [11]. Benefiting from the services provided by these intelligent applications and services, our lifestyles have been dramatically changed.

However, existing intelligent applications are computation-intensive, which present strict requirements on resources, e.g., CPU, GPU, memory, and network, and makes it impossible to be available anytime and anywhere for end users. Although current end devices are increasingly powerful, it is still insufficient to support some deep learning models. For example, most voice assistants, e.g., Apple Siri Google Assistant and Microsoft's Cortana, are based on cloud computing and they can not work if the network is unavailable. Moreover, existing intelligent applications generally adopt centralised data management, which requires users to upload their data to central data-centre. However, there is giant volume of data, generated and collected by billions of mobile users and Internet of Thing (IoT) devices, which are distributed at the network edge. According to Cisco's forecast, there will be 850 ZB of data generated by mobile users and IoT devices by 2021 [12]. Uploading such volume of data to the cloud consumes significant bandwidth resources, which may also result in unacceptable latency for users [13]. On the other hand, users increasingly concern about their privacy issues. The European Union has promulgated General Data Protection Regulation (GDPR) to protect private information of users [14]. If mobile users upload their personal data to the cloud for a specific intelligent application, they would take the risk of privacy leakage, i.e., the personal data might be extracted by malicious hackers or companies for illegal purposes.

Edge computing [15]–[20] emerges as an extension of cloud computing to push cloud services to the proximity of end users. Edge computing offers computing platforms which provide computing, storage, and networking resources, which are usually located at the edge of networks. The devices that provide services for end devices are referred to as edge servers, which could be IoT gateways, routers, and micro data centers at mobile network base stations, on vehicles, and amongst other places. End devices, such as mobile phones, IoT devices, and embedded devices that request services from edge servers are called edge devices. With the fast development of end devices, the capabilities of computing and energy control has been significantly improved, which makes it possible to provide networking and lightweight computing services for peers [21]–[25]. The main advantages of the edge computing paradigm could be summarised into three aspects. (i) Ultra-low latency: computation usually takes place in the proximity of the source data, which saves substantial amounts of time

D. Xu, T. Li, S. Tarkoma and P. Hui are with the Department of Computer Science, University of Helsinki, 00014 Helsinki, Finland.(e-mail: dianlei.xu@helsinki.fi, t.li@connect.ust.hk, sasu.tarkoma@helsinki.fi, panhui@cse.ust.hk.)

D. Xu, Y. Li and T. Li are with Beijing National Research Center for Information Science and Technology (BNRist), Department of Electronic Engineering, Tsinghua University, Beijing 100084, China.(e-mail: liyong07@tsinghua.edu.cn.)

T. Li and P. Hui are also with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong.

X. Su is with the Department of Computer Science, Norwegian University of Science and Technology, Norway and Center of Ubiquitous Computing, University of Oulu, Finland. (e-mail: xiang.su@ntnu.no)

T. Jiang is with the School of Electronics Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China. (e-mail: taojiang@ieee.org)

J. Crowcroft is with the Computer Laboratory, University of Cambridge, William Gates Building, 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK. (e-mail: Jon.Crowcroft@cl.cam.ac.uk.)

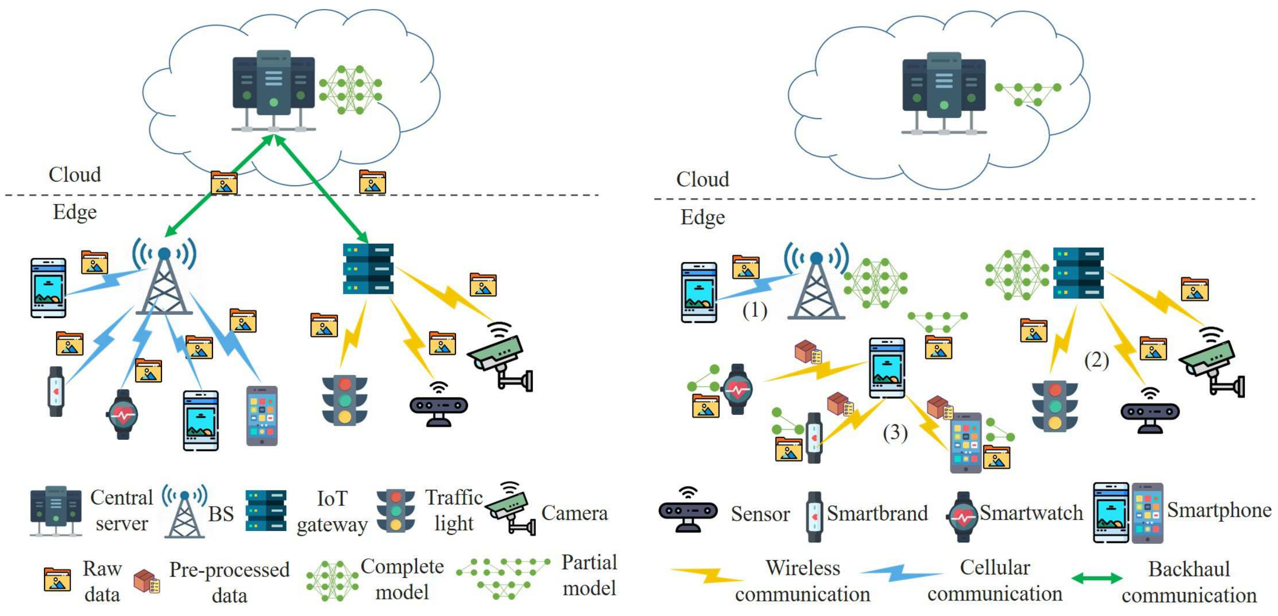
on data transmission. Edge servers provides nearly real-time responses to end devices. (ii) Saving energy for end devices: since end devices could offload computing tasks to edge servers, the energy consumption on end devices would significantly shrink. Consequently, the battery life of end devices would be extended. (iii) Scalability: cloud computing is still available if there are no enough resource on edge devices or edge servers. In such a case, the cloud server would help to perform tasks. In addition, end devices with idle resources could communicate amongst themselves to collaboratively finish a task. The capability of the edge computing paradigm is flexible to accommodate different application scenarios.

Edge computing addresses the critical challenges of AI based applications and the combination of edge computing and AI provides a promising solution. This new paradigm of intelligence is called edge intelligence [26], also named mobile intelligence [27]. Edge intelligence refers to a set of connected systems and devices for data collection, caching, processing, and analysis proximity to where data is collected, with the purpose of enhancing the quality and speed of data processing and to protect the privacy and security of data. Compared with traditional cloud-based intelligence that requires end devices to upload generated or collected data to the remote cloud, edge intelligence processes and analyses data locally, which effectively protects users' privacy, reduces response time, and saves on bandwidth resources [28], [29]. Moreover, users could also customise intelligent applications by training ML/DL models with self-generated data [30], [31]. It is predicted that edge intelligence will be a vital component in 6G network [32]. It is also worth noting that AI could also be a powerful assistance for edge computing. This paradigm is called intelligent edge [33], which is different from edge intelligence. The emphasis of edge intelligence is to realize intelligent applications in edge environment with the assistance of edge computing and protect users' privacy, while intelligent edge focuses on solving problems of edge computing with AI solutions, e.g., resource allocation optimization. Intelligent edge is out of our scope in this survey.

There exists lots of works which have proved the feasibility of edge intelligence by applying an edge intelligence paradigm to practical applications. Yi *et al.* implement a face recognition application across a smartphone and edge server [34]. Results show that the latency is reduced from 900ms to 169ms, compared with cloud based paradigm. Ha *et al.* use a cloudlet to help a wearable cognitive assistance execute recognition tasks, which saves energy consumption by 30%-40% [35]. Some researchers pay attention to the performance of AI in the context of edge computing. Lane *et al.* successfully implement a constrained DL model on smartphones for activity recognition [36]. The demo achieves a better performance than shallow models, which demonstrates that ordinary smart devices are qualified for simple DL models. Similar verification is also done on wearable devices [37] and embedded devices [38]. The most famous edge intelligence application is Google G-board, which uses federated learning [39] to collaboratively train the typing prediction model on smartphones. Each user uses their own typing records to train G-board. Hence, the trained G-board could be used immediately, powering experiences personalised by the way users use this application.

This paper aims at providing a comprehensive survey to the development and the state-of-the-art of edge intelligence. As far as we know, there exist few recent efforts [40]–[45] in this direction, but they have very different focuses from our survey. Table I summarizes the comparison among these works. Specifically, Yang *et al.* provide a survey on federated learning, in which they mainly focus on the architecture and applications of federated learning [40]. The authors divide literature of federated learning into three classifications: horizontal federated learning, vertical federated learning, and federated transfer learning. Shi *et al.* pay attention to the wireless communication overhead in federated learning and survey algorithms and systems on decreasing the wireless communication cost in this area [41]. Horizontal federated learning and communication-efficient federated learning are involved as a cloud-based federated learning in our survey. We pay more attention to the combination of federated learning and edge computing, i.e., edge-based federated learning and hierarchical federated learning, in addition to cloud-based federated learning. The focus of [42] is how to realize the training and inference of DL models on a single mobile device. They briefly introduce some challenges and existing solutions from the perspective of training and inference. By contrast, we provide a more comprehensive and deeper review on solutions from the perspective of model design, model compression, and model acceleration. We also survey how to realize model training and inference with collaboration of edge devices and edge servers, even the assistance from the cloud server, in addition to solo training and inference at edge.

To our best knowledge, Ref. [43], [44] and [45] are three most relevant articles to our survey. The focus of Ref. [43] is the inter-availability between edge computing and DL. Hence the scope of Ref. [43] includes two parts: DL for edge computing, and edge computing for DL. The former part focuses on some optimisation problems at edge with DL approaches, for example, caching popular content and resource allocation in wireless network with deep reinforcement learning, which is out of our scope. The latter part focus on applying DL in the context of edge computing, which is overlapping to our survey to some extent. Specifically, in this part, they mainly review the enablers of using DL in the context of edge, e.g., the hardware, theory, and methods. The scope and content of Ref. [44] are highly overlapped with [43]. The survey [45] mainly focuses on reviewing and discussing the development, motivation, architecture, and enabling theories of edge intelligence. We also discuss the architectures and enabling theories. For the overlapped part with our paper, these three surveys only discuss *what enables edge intelligence* without discussing how these enablers are used and their performance in various kinds of scenarios. In contrast, we adopt an orthogonal view to review *how to enable edge intelligence*. For example, in edge training, they only introduce the theory of cloud-based federated learning. Differently, we review how edge-based and hierarchical federated learning are used in various application scenarios, as well as their strong points and shortcomings, in addition to cloud-based federated learning. We also discuss



(a) Centralized intelligence

(b) Edge intelligence

Fig. 1. The comparison of traditional intelligence and edge intelligence from the perspective of implementation. In traditional intelligence, all data must be uploaded to a central cloud server, whilst in edge intelligence, intelligent application tasks are done at the edge with locally-generated data in a distributed manner.

the privacy and security problems in federated learning and how these problems are utilized by hackers, instead of briefly mentioning what they are. In edge inference, they all generally introduce model compression and acceleration theories. We review how these theories are associated with practical applications with a deeper and finer-grained view. we summarize how to realize model compression and their feasibility with various approaches in different scenarios, as well as how to accelerate inference from the view of hardware and software, respectively. Besides these overlaps, we also investigate other key components in implementing AI in edge environment, i.e., edge data collection/management, and model offloading, which are not included in other surveys.

It is worth noting that, the edge caching and offloading in our paper is different from these extensively studied traditional caching and offloading. For edge caching, we review works of caching AI models at edge to deal with task-fickle scenarios. Pre-caching multiple kinds of deep learning models on edge server for different kinds of tasks can reduce the computation time and further improve users' QoE in edge intelligence. We also review works on caching computation results of AI models, which are reusable for future execution of AI applications on edge devices. Both of these two aspects are very relevant and important to the implementation of edge intelligence and naturally different from traditional edge caching for content download/delivery. For edge offloading in our survey, we mainly focus on offloading tasks of AI applications with full consideration of the characteristics of AI models, instead of purely computing.

Our survey focuses on how to realise edge intelligence in a systematic way. There exist three key components in AI, i.e. data, model/algorithm[1], and computation. A complete process of implementing AI applications involves data collection and management, model training, and model inference. Computation plays an essential role throughout the whole process. Hence, we limit the scope of our survey on four aspects, including how to cache data to fuel intelligent applications (i.e., edge caching), how to train intelligent applications at the edge (i.e., edge training), how to infer intelligent applications at the edge (edge inference), and how to provide sufficient computing power for intelligent applications at the edge (edge offloading). Our contributions are summarized as following:

- We survey recent research achievements on edge intelligence and identify four key components: edge caching, edge training, edge inference, and edge offloading. For each component, we outline a systematical and comprehensive classification from a multi-dimensional view, e.g., practical challenges, solutions, optimisation goals, etc.
- We present thorough discussion and analysis on relevant papers in the field of edge intelligence from multiple views, e.g., applicable scenarios, methodology, performance, etc. and summarise their advantages and shortcomings.
- We discuss and summarise open issues and challenges in the implementation of edge intelligence, and outline five important future research directions and development trends, i.e., data scarcity, data consistency, adaptability of model/algorithms, privacy and security, and incentive mechanisms.

The remainder of this article is organized as follow. Section II overviews the research on edge intelligence, with considerations of the essential elements of edge intelligence, as well

[1]Model and algorithm are interchangeable in this article

TABLE I
COMPARISON OF RELATIVE SURVEYS.

Ref.	Year	Domain	Scope
[40]	2019	Federated learning	Horizontal federated learning, vertical federated learning, and federated transfer learning
[41]	2020	Federated learning	Wireless communication overhead
[42]	2018	DL-based mobile applications	Training and inference on single mobile device
[43]	2019	Edge intelligence Intelligent edge	Enablers of training and inference at edge Optimizing edge problems with DL
[44]	2020	Edge intelligence Intelligent edge	Enablers of training and inference at edge Optimizing edge problems with DL
[45]	2019	Edge intelligence	Development, motivation Architectures and enablers
Our work	2021	Edge intelligence	Architectures, challenges, enablers, applicable scenarios Performance analysis, edge caching and edge offloading

as the development situation of this research field. Then, we give a brief review on artificial intelligence in Section III. We present detailed introduction, discussion, and analysis on the development and recent advances of edge caching, edge training, edge inference, and edge offloading in Section IV to Section VII, respectively. Finally, we discuss the open issue and possible solutions for future research in Section VIII, and conclude the paper in Section IX.

II. Overview

For convenience, we present the comparison between traditional centralised intelligence with edge intelligence from the perspective of implementation in Fig. 1. Traditional centralised intelligence is shown in Fig. 1(a), where all edge devices first upload data to the central server for intelligent tasks, e.g., model training or inference. The central server/data-centre is usually, but not necessarily, located in remote cloud. After the processing on the central server, results, e.g., recognition or prediction results, are transmitted back to edge devices. Fig. 1(b) demonstrates the implementation of edge intelligence, where a task, e.g., recognition or prediction, is either done by edge servers and peer devices, or with the edge-cloud cooperation paradigm. A very small amount, or none of the data is uploaded to the cloud. For example, in area (1) and (2), cloudlet, i.e. BS and IoT gateway could run complete intelligent models/algorithms to provide services for edge devices. In area (3), a model is divided into several parts with different functions, which are performed by several edge devices. These edge devices work together to finish the task.

It is known that three most important elements for an intelligent application are: data, model, and computation. Suppose that an intelligent application is a 'human', model would be the 'body', and computation is the 'heart' which powers the 'body'. Data is then the 'book'. The 'human' improves their abilities by learning knowledge extracted from the 'book'. After learning, the 'human' starts to work with the learned knowledge. Correspondingly, the complete deployment of most intelligent applications (unsupervised learning based application is not included) includes three components: data collection and management (preparing the 'book'), training (learning), and inference (working). Computation is a hidden component that is essential for the other three components. Combined with an edge environment, these three obvious components turn into edge cache (data collection and storage at edge), edge training (training at edge), and edge inference (inference at edge), respectively. Note that edge devices and edge servers are usually not powerful. Computation at edge usually is done via offloading. Hence, the hidden component turns into edge offloading (computation at edge). Our classification is organised around these four components, each of which features multidimensional analysis and discussion. The global outline of our proposed classification is shown in Fig. 2. For each component, we identify key problems in practical implementation and further break down these problems into multiple specific issues to outline a tiered classification. Next, we present an overview of these modules shown as Fig. 2.

A. Edge Caching

In edge intelligence, edge caching refers to a distributed data system proximity to end users, which collects and stores the data generated by edge devices and surrounding environments, and the data received from the Internet to support intelligent applications for users at the edge. Fig. 3 presents the essential idea of edge caching. Data is distributed at the edge. For example, mobile users' information generated by themselves is stored in their smartphones. Edge devices such as monitoring devices and sensors record the environmental information. Such data is stored at reasonable places and used for processing and analysis by intelligent algorithms to provide services for end users. For example, the video captured by cameras could be cached on vehicles for aided driving [46]. Edge caching is different from traditional caching, which caches popular content at base stations from the backbone network. The data flow of traditional caching is from the cloud to the edge of the network, while the data flow of edge caching is from the edge to the central cloud. The structure of this section is organised as the bottom module in Fig. 2.

In edge caching, there are two kinds of content to be cached at edge. One is the raw data collected from end users or IoT devices and surrounding environment, which provides the input for intelligent applications. The inputs of an intelligent application may be the same or partially the same. For example, in continuous mobile vision analysis, there are large amounts of similar pixels between consecutive frames. Some resource-constrained edge devices need to upload collected videos to edge servers or the cloud for further processing. With cache, edge devices only need to upload different pixels or frames. For the repeated part, edge devices could reuse the

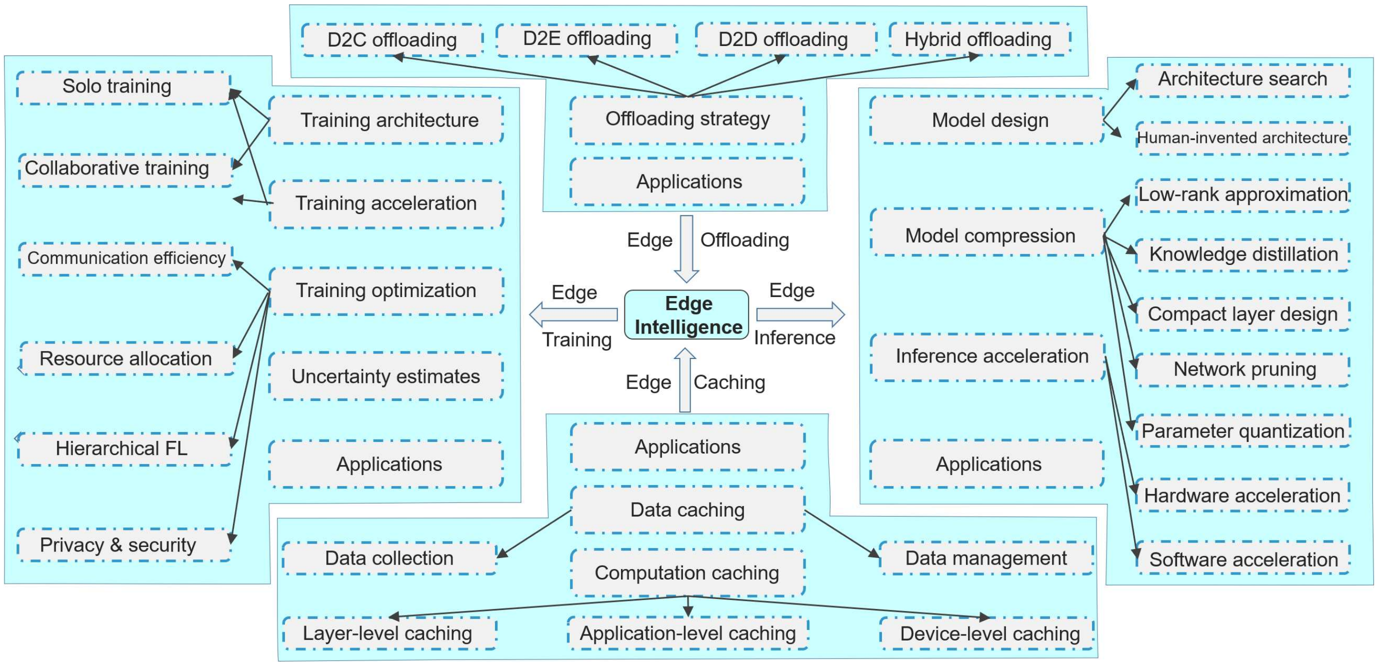


Fig. 2. The classification of edge intelligence literature.

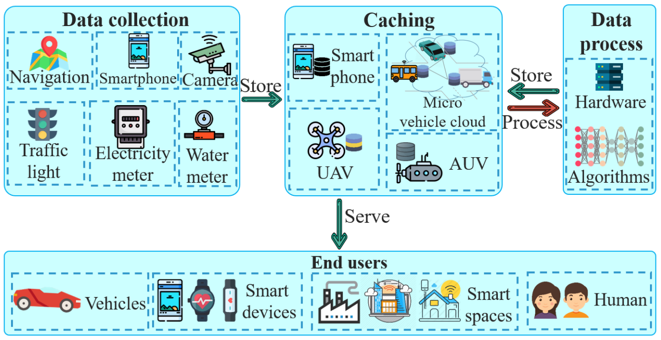


Fig. 3. The illustration of edge caching. Data generated by mobile users and gathered from surrounding environments is collected and stored on edge devices, unmanned aerial vehicle (UAV), autonomous underwater vehicles (AUV), and micro clouds. Such data is processed and analysed by intelligent algorithms to provide services for end users.

results to avoid unnecessary computation. Hence, the other kind of cached data is the computation results of some intelligent applications, which could be used in the future to avoid redundant computation. Moreover, the requested computing tasks of intelligent applications may be the same. For example, an edge server provides image recognition services for edge devices. The recognition tasks from the same context may be the same, e.g., the same tasks of flower recognition from different users of the same area. Edge servers could directly send the recognition results achieved previously back to users. Such kind of caching could significantly decrease computation and execution time. Some practical applications based on computation redundancy are developed in [47]–[49].

For data caching, existing works mainly focus on how to efficiently collect data and how to manage the collected data. Mobile users could keep their data locally on their smartphones for privacy and security concern. However, the majority data is generated by IoT devices, which have extremely limited resources. Hence, the collected data need to be sent to the edge data center, e.g., edge server. Some researchers focus on directly collecting data through cellular network [50]–[69]. These devices are equipped with SIM card, so that they can directly upload sensing data to data center. In addition, some researchers pay attention to data collection with opportunistic networks in urban with the assistance of vehicles [54], [70]–[82]. These vehicles can collect data from IoT devices when they pass by or directly sense data from the urban environment. However, not all IoT devices are implemented near the road. Some researchers adopts multi-hop routing for data delivery [83]–[86]. There are also some work collecting data in underwater scenarios for smart ocean [87], [88], [88]–[91].

In edge context, the collected data is kept locally for processing and maintaining. Different from cloud-based centralized data center, there is no large-scale data center at edge. Existing works mainly focus on storing and managing data on edge data repository [92]–[99] and local edge device, especially micro cloud [100], [101], [101]–[111]. In addition, we also give some practical applications with these data collection and management techniques [112]–[114].

Researchers have verified the redundant computation in AI applications [115], [116]. Caching reusable computation results could significantly reduce the computation latency and energy consumption. According to the caching granularity, we identify three kinds of computation caching: level-layer caching, application-level caching, and device-level caching. For level-layer caching, existing works mainly focus on the similarity-based lookup and incremental learning on changed

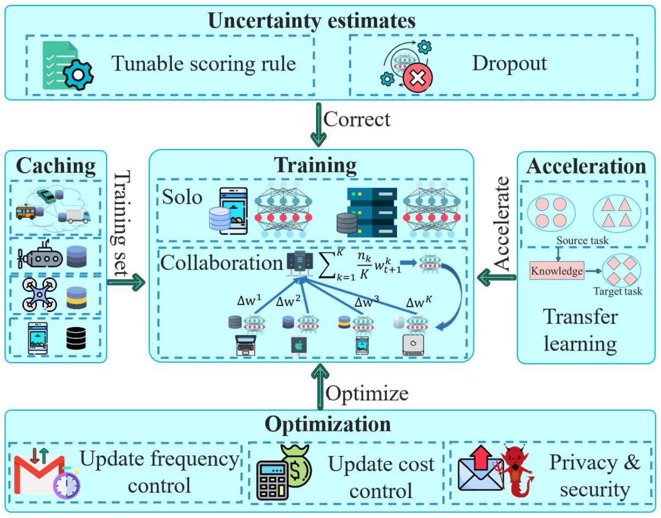


Fig. 4. The illustration of edge training. The model/algorithm is trained either on a single device (solo training), or by the collaboration of edge devices (collaborative training) with training sets cached at the edge. Acceleration module speeds up the training, whilst the optimisation module solves problems in training, e.g., update frequency, update cost, and privacy and security issues. Uncertainty estimates module controls the uncertainty in training.

pixels in continuous mobile vision applications [46], [117]–[125]. For application-level caching, the focus of literature is approximate caching across applications on the same device [126]–[129]. For device-level caching, researchers investigate the spatio-temporal locality of users within an area to cache repeatedly requested computation results on edge server [47]–[49], [130]–[133]. In addition, there are also some works propose to cache multiple deep learning models on edge server for specialized missions to improve the quality of service [134]–[137]. Some applications on eye gaze tracking [138] and voice assistant [139], [140] based on computation caching are introduced at last.

B. Edge Training

Edge training refers to a distributed learning procedure that learns the optimal values for all the weights and bias, or the hidden patterns based on the training set cached at the edge. For example, Google develops an intelligent input application, named G-board, which learns user's input habits with the user's input history and provides more precise prediction on the user's next input [39]. The architecture of edge training is shown as Fig. 4. Different from traditional centralised training procedures on powerful servers or computing clusters, edge training usually occurs on edge servers or edge devices, which are usually not as powerful as centralised servers or computing clusters. Hence, in addition to the problem of training set (caching), four key problems should be considered for edge training: (i) how to train (the training architecture), (ii) how to make the training faster (acceleration), (iii) how to optimise the training procedure (optimisation), and (iv) how to estimate the uncertainty of the model output (uncertainty estimates). The structure of this section is organised as the left module in Fig. 2.

For the first problem, researchers design two training architectures: solo training [36]–[38], [141] and collaborative training [39], [142]–[145]. Solo training means training tasks are performed on a single device, without assistance from others, whilst collaborative training means that multiple devices cooperate to train a common model/algorithm. Since solo training has a higher requirement on the hardware, which is usually unavailable, most existing literature focuses on collaborative training architectures.

Different from centralised training paradigms, in which powerful CPUs and GPUs could guarantee a good result with a limited training time, edge training is much slower. Some researchers pay attention to the acceleration of edge training. Corresponding to training architecture, works on training acceleration are divided into two categories: acceleration for solo training [141], [146]–[151], and collaborative training [145], [152], [153].

Solo training is a closed system, in which only iterative computation on single devices is needed to get the optimal parameters or patterns. In contrast, collaborative training is based on the cooperation of multiple devices, which requires periodic communication for updating. Update frequency and update cost are two factors which affect the performance of communication efficiency and training result. Researchers on this area mainly focus on how to maintain the performance of the model/algorithm with lower update frequency [154], [154], [155], [155]–[165], and update cost [152], [155], [166]–[169]. To solve the problem of limited resources, e.g., CPU and battery, researchers propose various energy-efficiency algorithms to help edge devices collaboratively train models under various kinds of scenarios [170]–[178]. In addition, totally edge-based collaborative training is limited the scarcity of participants and data. Some researchers adopt hierarchical federated learning-based approaches to solve such problem [179]–[184].

The public nature of collaborative training is vulnerable to malicious users. There is also some literature which focuses on the privacy [156], [185]–[195] and security [196]–[200], [200]–[202], [202]–[205] issues.

In DL training, the output results may be erroneously interpreted as model confidence. Estimating uncertainty is easy on traditional intelligence, whilst it is resource-consuming for edge training. Some literature [206], [207] pays attention to this problem and proposes various kinds of solutions to reduce computation and energy consumption.

We also summarised some typical applications of edge training [39], [154]–[156], [174], [208]–[212], [212]–[223] that adopt the above-mentioned solutions and approaches.

C. Edge Inference

Edge inference is the stage where a trained model/algorithm is used to infer the testing instance by a forward pass to compute the output on edge devices and servers. For example, developers have designed a face verification application based DL, and employ on-device inference [224], [225], which achieves high accuracy and low computation cost. The architecture of edge inference is shown as Fig. 5. Most existing AI models are designed to be implemented on devices which

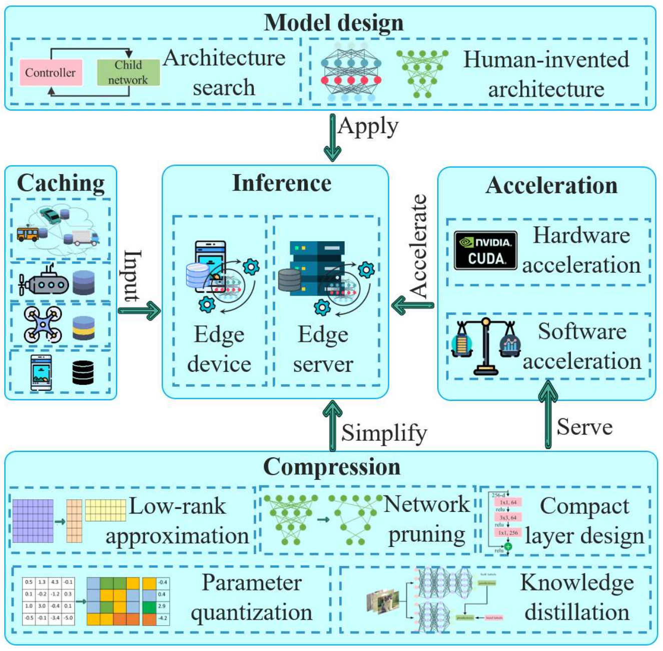


Fig. 5. The illustration of edge inference. AI models/algorithms are designed either by machines or humans. Models could be further compressed through compression technologies: low-rank approximation, network pruning, compact layer design, parameter quantisation, and knowledge distillation. Hardware and software solutions are used to accelerate the inference with input data.

have powerful CPUs and GPUs, this is not applicable in an edge environment. Hence, the critical problems of employing edge inference are: (i) how to make models applicable for their deployment on edge devices or servers (design new models, or compress existing models), and (ii) how to accelerate edge inference to provide real-time responses. The structure of this section is organised as the right module in Fig. 2.

For the problem of how to make models applicable for the edge environment, researchers mainly focus on two research directions: design new models/algorithms that have less requirements on the hardware, naturally suitable for edge environments, and compress existing models to reduce unnecessary operation during inference. For the first direction, there are two ways to design new models: let machines themselves design optimal models, i.e., architecture search [226], [227], [227], [227]–[230]; and human-invented architectures with the application of depth-wise separable convolution [231]–[233] and group convolution [234], [235]. We also summarise some typical applications based on these architectures, including face recognition [224], [225], [236], human activity recognition (HAR) [237]–[245], vehicle driving [246]–[249], and audio sensing [250], [251]. For the second direction, i.e., model compression, researchers focus on compressing existing models to obtain thinner and smaller models, which are more computation- and energy-efficient with negligible or even no loss on accuracy. There are five commonly used approaches on model compression: low-rank approximation [252]–[257], knowledge distillation [258]–[266], compact layer design [267]–[275], network pruning [276]–[290], and parameter quantisation [253], [290]–[306]. In addition, we

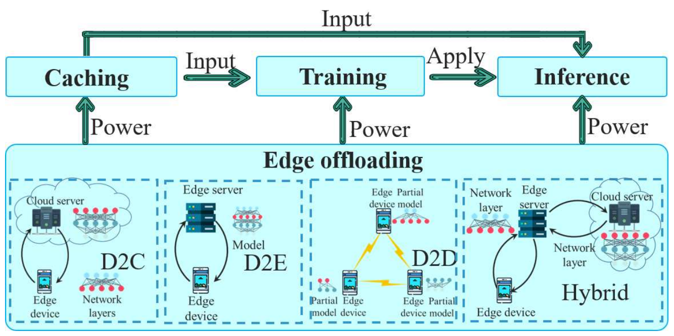


Fig. 6. The illustration of edge offloading. Edge offloading is located at the bottom layer in edge intelligence, which provides computing services for edge caching, edge training, and edge inference. The computing architecture includes D2C, D2E, D2D, and hybrid computing.

also summarise some typical applications [307]–[313] that are based on model compression.

Similar to edge training, edge devices and servers are not as powerful as centralised servers or computing clusters. Hence, edge inference is much slower. Some literature focuses on solving this problem by accelerating edge inference. There are two commonly used acceleration approaches: hardware acceleration and software acceleration. Literature on hardware acceleration [118], [314]–[323], [323]–[338] mainly focuses on the parallel computing which is available as hardware on devices, e.g., CPU, GPU, and DSP. Literature on software acceleration [46], [118], [119], [339]–[346] focus on optimising resource management, pipeline design, and compilers, based on compressed models.

D. Edge offloading

As a necessary component of edge intelligence, edge offloading refers to a distributed computing paradigm, which provides computing service for edge caching, edge training, and edge inference. If a single edge device does not have enough resource for a specific edge intelligence application, it could offload application tasks to edge servers or other edge devices. The architecture of edge offloading is shown as Fig. 6. Edge offloading layer transparently provides computing services for the other three components of edge intelligence. In edge offloading, Offloading strategy is of utmost importance, which should give full play to the available resources in edge environment. The structure of this section is organised as the top module in Fig. 2.

Available computing resources are distributed in cloud servers, edge servers, and edge devices. Correspondingly, existing literature mainly focuses on four strategies: device-to-cloud (D2C) offloading, device-to-edge server (D2E) offloading, device-to-device (D2D) offloading, and hybrid offloading. Works on the D2C offloading strategy [125], [347]–[361] prefer to leave pre-processing tasks on edge devices and offload the rest of the tasks to a cloud server, which could significantly reduce the amount of uploaded data and latency. Works on D2E offloading strategy [362]–[369] also adopt such operation, which could further reduce latency and the dependency on cellular network. Most works on D2D offloading

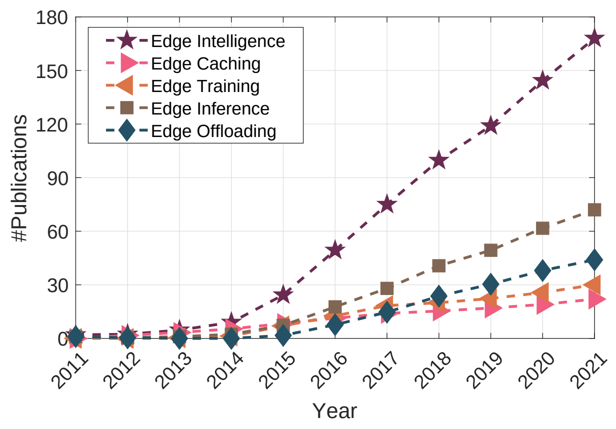


Fig. 7. Publication volume over time. These curves show the trend of publication volume in edge caching, edge training, edge computing, edge inference, and edge intelligence, respectively. The value of 2021 is predicted based on the values of past years.

strategy [370]–[377] focus on smart home scenarios, where IoT devices, smartwatches and smartphones collaboratively perform training/inference tasks. Hybrid offloading schemes [378]–[380] have the strongest ability of adaptiveness, which makes the most of all the available resources.

We also summarise some typical applications that are based on these offloading strategies, including face recognition [381], intelligent transportation [382], smart industry [383], smart city [384], and healthcare [385]–[387].

E. Summary

In our survey, we identify four key components of edge intelligence, i.e. edge caching, edge training, edge inference, and edge offloading. Edge intelligence shows an explosive developing trend with a huge amount of researcher have been carried out to investigate and realise edge intelligence over the past five years. We count the publication volume of edge intelligence, as shown in Fig. 7.

We see that this research area started from 2011, then grew at a slow pace before reaching 2014. Most strikingly, after 2014, there is a rapid rise in the publication volume of edge caching, edge training, edge inference, and edge offloading. Overall, the publication volume of edge intelligence is booming, which demonstrates a research field replete with activity. Such prosperity of this research filed owes to the following three reasons.

First, it is the booming development of intelligent techniques, e.g., deep learning and machine learning techniques that provides a theoretical foundation for the implementation of edge intelligence [388]–[390]. Intelligent techniques achieve state-of-the-art performance on various fields, ranging from voice recognition, behaviour prediction, to automatic piloting. Benefiting from these achievements, our life has been dramatically changed. People hope to enjoy smart services anywhere and at any time. Meanwhile, most existing intelligent services are based on cloud computing, which brings inconvenience for users. For example, more and more people are using voice assistant on smartphone, e.g., MI AI and Apple Siri. However, such applications can not work without networks.

Second, the increasing big data distributed at the edge, which fuels the performance of edge intelligence [391]–[393]. We have entered the era of IoT, where a giant amount of IoT devices collect sensory data from surrounding environment day and night and provide various kinds services for users. Uploading such giant amount of data to cloud data centres would consume significant bandwidth resources. Meanwhile, more and more people are concerned about privacy and security issues behind the uploaded data. Pushing intelligent frontiers is a promising solution to solve these problems and unleash the potential of big data at the edge.

Third, the maturing of edge computing systems [16], [18] and peoples' increasing demand on smart life [394], [395] facilitate the implementation of edge intelligence. Over the past few years, the theories of edge computing have moved towards application, and various kinds of applications have been developed to improve our life, e.g., augmented reality [396]–[398]. At the same time, with the wide spreading of 5G networks, more and more IoT devices are implemented to construct smart cities. People are increasingly reliant on the convenient service provided from a smart life. Large efforts from both academia and industry are enacted to realise these demands.

III. Preliminary of Artificial Intelligence

In this section, we briefly introduce the concept and its branches of artificial intelligence.

A. Artificial Intelligence

The history of artificial intelligence could be traced back to 1956. After decades of development, it has been the hottest topic in both academia and industry. Artificial intelligence refers to the ability enabling computers to mimic intelligent creatures, which learn from experience to adapt to various kinds environment and tasks [399]–[401]. The goal of AI is to let computer act like human to understand and think and replace human in various kinds of tasks, such as object recognition, translation, and driving.

The scope of artificial intelligence is quite wide, including knowledge representation, reasoning, data mining, etc. Learning is an approach to enable computer to be intelligent. The relationship amongst artificial intelligence, machine learning and deep learning is shown as Fig. 8. Machine learning is a typical approach to realize artificial intelligence, while deep learning is an efficient machine learning algorithm.

B. Machine learning

Machine learning refers to the process of computer learning and improving with knowledge extracted from data, without following specific instructions [402]. Generally, there are three branches of machine learning: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is applicable to scenarios where data is labeled. Supervised

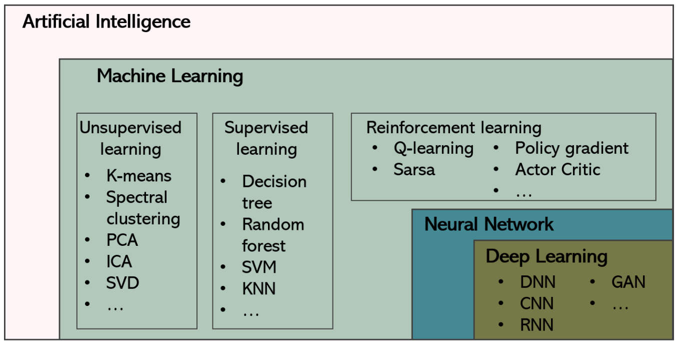


Fig. 8. Publication volume over time. These curves show the trend of publication volume in edge caching, edge training, edge computing, edge inference, and edge intelligence, respectively.

learning algorithms learn a function which maps the input to output. The learning procedure is to continuously minimize the loss function that reflects the overall degree of input-output matching. Popular supervised algorithms include decision tree [403], random forest [404], support vector machine (SVM) [405], k-nearest neighbors (KNN) [406], linear regression [407], etc. In unsupervised learning, the data is not labeled. Learning algorithms automatically learn the properties of data. Typical examples of unsupervised learning include grouping data items based on similarity, and reducing dimension of data. Popular unsupervised learning algorithms include K-means [408], spectral clustering [409], principle component analysis (PCA) [410], independent component analysis (ICA) [411], singular value decomposition (SVD) [412], etc. Similar with unsupervised learning, there is also no labeled data in reinforcement learning. Intelligent agents continuously improve their actions to maximize the cumulative rewards through the interaction with the surrounding environment. For example, if the agent plays chess, it can try different strategies of action. If the strategy can help the agent get more points, higher reward would be given to this strategy. Typical algorithms of reinforcement learning include Q-learning [413], double Q-learning [414], Sarsa [415], deep Q-network [416], policy gradient [417], actor critic [418], etc.

Generally, machine learning requires to extract features manually, which is also called feature engine. This is feasible in some simple scenarios. However, in scenarios like object detection and voice recognition, it is hard to design features by human.

C. Deep learning

Deep learning evolves from neural network (NN), which is a simulation of bio neural network. Neuron is the basic element of neural network, which receives input from other neurons (except for the input layer), process received inputs with activation function, and output the results to next neurons. In deep neural network (DNN), there are multiple layers and each layer has multiple neurons. In particular, there is one input layer, one output layer, and multiple hidden layers between input layer and output layer. Data is sent to the input layer, which will further propagate the data to hidden layer. Then the data is processed and propagate through multiple hidden layers and finally the results are output through output layer. The most obvious difference between DNN and traditional NN is the number of hidden layers, which automatically extracts feature with non-linear units. There are usually multiple hidden layers in DNN, which is also the origin of ”deep learning”.

Theoretically, deep learning is able to approximate any continuous function [419], [420]. With the network going deeper, it can solve a lot of complex problems. Practically, deep learning has achieved state-of-the-art performance in various kinds application scenarios, which proves the learning ability of deep learning. Deep learning highly relies on the data. With more high-quality data, the performance of deep learning will be better. However, more data and deeper model mean more computing power. Most deep learning based applications are not suitable to be executed on resource-constraint edge devices. Hence, the scope of our survey is how to implement applications/models in the context of edge computing.

Typical models of deep learning include convolutional neural network (CNN) [421], recurrent neural network (RNN) [422], generative adversarial network (GAN) [423], and their variations. Specifically, CNN is often used for image processing, e.g., image classification and face recognition. The structure of CNN is usually composed of convolutional layers, pooling layers, and fully connected layers. Convolutional layer is used to extract features from input data, while pooling layer is used to down sampling feature maps and reduce the dimension. The fully connected layer is a classifier to output final result. RNN is usually used to process sequential data, e.g., time series and language. The most obvious characteristic of sequential data is the temporal-correlation. The structure of RNN is also composed of input layer, hidden layer, and output layer. RNN allows neurons in hidden layer to keep the memory of previous input, which will impact the processing result of current input. For example, in stock prediction, the current price is high related to the price of yesterday, even the price of last week. However, RNN has the shortcoming of short-term memory. Hence, variations, i.e., long short-term memory networks (LSTM) [424] and gated recurrent units (GRU) [425] are proposed to solve the problem. GAN is composed of two parts: generator and discriminator. The generator is used to generate data according to the learned distribution of data, while the discriminator is used to judge on result of the generator. The training procedure is as follows. (1) Train the generator and use the discriminator to judge the results of generator, until the discriminator cannot tell the generated result from real data. (2) Train the discriminator until it is able to correctly judge the generated results. (3) Repeat step (1) and (2). Finally, we get a generator and a discriminator with good performance.

IV. Edge Caching

Initially, the concept of caching comes from computer systems. The cache was designed to fill the throughput gap between the main memory and registers [426] by exploring correlations of memory access patterns. Afterwards, the caching idea was introduced in networks to fill the throughput gap between core networks and access networks. Nowadays, the cache is widely deployed at edge of the network, such as various base stations and edge devices. By leveraging the spatiotemporal redundancy of communication and computation tasks, caching at the edge can significantly reduce transmission and computation latency and improve QoE of users [427]–[429]. From existing studies, the critical issues of caching technologies in edge networks fall into two aspects, including data caching and computation caching. Next, we discuss and analyse relevant literature in edge caching in terms of these two perspectives.

A. Data caching

DL algorithms, e.g., CNN and RNN, are able to automatically learn patterns and extract features from input data without the intervention of human [430], [431]. Generally, to obtain an algorithm with high performance, especially algorithms with large amount of parameters, giant volume of data and computing operation are required during the learning process [432]–[435]. On the other hand, billions of end devices, e.g., IoT devices and smartphones, are deployed at the edge of the network to sense and collect the status and information of the crowd flow, environment and infrastructures. There are mainly two problems to utilize the data generated by these end devices to realize DL algorithms. The first problem is the "data island" phenomenon for separate end devices. The data generated or collected by a single end device is inadequate to obtain a well-performing DL algorithm. The second problem is the limited computing and storage capability of end devices. Although current smartphones are powerful, which are able to train simple DL algorithms, most end devices are resource-constraint IoT devices, e.g., sensors, which are not able to train DL algorithms separately.

Hence, edge data caching naturally appears, aiming at gathering sufficient data and cache it at edge data centers [436], [437] with sufficient storage and computing resource, e.g., edge server, powerful edge devices, etc., to solve aforementioned problems. Traditional data caching mainly focuses on the storage system of popular files or document on edge servers that is physically close to edge devices to meet the requirements of latency and bandwidth [438]–[440]. The data flow of traditional data caching is from the core of the network to the edge [441], [442]. In contrast, data caching focuses on the sharing, collection, storage and management of the data generated at the edge of the network to provide better data service for intelligent applications [443]. The data flow is from the edge of the network to the core [92].

Edge data caching can provide massive amount of training data for edge intelligent applications. For example, Jeong *et al.* develop the City Data Hub, a smart city platform, to manage the cached sensing data in the city [444]. The system provides API for various kinds of data-driven applications, e.g., COVID-19 Investigation System, which requires massive amount of tracing data for training to provide accurate prediction and detection service. In addition to model training, model inference also requires data input. However, most IoT sensors are not able to inference models. Hence, data is transmitted to and cached on edge server for inference. Soliman *et al.* implement wireless sensors in forest to collect massive amount of temperature, light, and smoke data and train a neural network-based forest-fire detection algorithm [445]. Muthur *et al.* design an industrial IoT data collection and training system, named SWaT for security-relevant research [446]. The system adopts the Ethernet-based ring topology. Sensors keep monitoring the water flow and transmitting data to the controller for caching. The cached data is used to train anomaly detection algorithms on controller side [447]–[449].

The key challenges of data caching are how to efficiently collect and manage the data. For the convenience of readers, we summarize existing literature on data caching in table II.

1) Data collection: Data collection refers to the process of gathering quantitative information, e.g., temperature and traffic, or qualitative information, e.g., anomaly or not, from end devices, e.g., sensors and embedded devices. According to the communication mode, research on data collection could be classified into two categories: cellular communication and opportunistic communication-based data collection. Multi-layer architecture is commonly adopted architecture in both categories [450]. There are usually two layers in cellular communication-based data collection architecture. The edge data center is at the top layer, which is usually associated with base station, while end devices equipped with cellular communication module are at the bottom layer. End devices directly communicate with edge data center through cellular network. In opportunistic communication-based data collection architecture, there is a middle layer between top layer and bottom layer, where mobile sinks are used to receive data from end devices through opportunistic contact and transmit data to edge data center. The mobile sink could be UAV, autonomous underwater vehicle, or vehicle on the road, which depends the practical applying scenario.

Smartphones are equipped with various kinds of sensors. Hence, smartphones are naturally suitable for data collection. In addition, some IoT devices are also equipped with SIM card [451], [452]. Data collection schemes with such end devices usually adopt cellular communication-based architecture, where end devices send sensing data to edge server for caching through cellular network. Then the cached data is used to train intelligent algorithms. For example, Kong *et al.* develop CrowdSwitch, a system to recommend the optimal switch among different cellular operators based on the collected LTE signal with the purpose of better networking experience [50], [51]. CrowdSwitch utilizes smartphones to sense and send the LTE signal strength and corresponding location to the server through cellular network. Through analyzing the sensed big data, they create a signal heat map, which is the core of the system. Then they recommend the optimal switch for users based on the signal heat map through predicting their mobility. Such framework could also be used constructing indoor floorplans. Elhamshary *et al.* find that users' certain activities are identifiable through some sensors on smartphone [52]. Based on such observation, they collect the sensing data of smartphones and train a classifier to infer the building semantics according to user's activities. Wang *et al.* construct a smart smoke detection system with narrow band IoT (NB-

IoT) devices [53]. Sensing stream is transmitted from NB-IoT sensors to servers for model training first and then real-time inference.

In large-scale outdoor unfriendly environment, UAVs with cellular communication module are efficient for data collection. Cao *et al.* adopt clustering strategy to divide the area into several part and use UAV to collect data [54]. In addition, they use ant colony algorithm [453] to optimize the flying path to minimize the energy consumption of UAV. Similar path optimization algorithm is also adopted in [55] and [56]. In addition, some researchers pay attention to the fairness issues in data collection, which is caused by the heterogeneity of environment and sensed data volume. Li *et al.* propose a speed-control scheme to solve the unfairness problem [57]. The flying speed of UAV is inversely proportional to the number of IoT devices in a cluster. In addition, they also propose a dichotomous algorithm to maximize the volume of collected data from each IoT device.

The cellular communication-based architecture is of high scalability and adaptability. Cellular communication enables direct transmission from data generators to data center, which is a highly efficient data collection method. Once the intelligent algorithms are trained with sufficient cached data, they could be used for delay-sensitive applications [454]. There are some literatures apply such architecture in intelligent transportation [58]–[63], air quality monitoring [64]–[66], and anomaly detection [67]–[69].

However, the cellular communication-based architecture brings additional cost. Transmitting the collected data requires additional cellular communication module equipped on end devices, which results in high hardware cost. Communication between end devices and base stations also requires spectrum resource to be assigned. Hence, cellular based architecture cannot be adopted in large scale data collection. Moreover, in areas of poor coverage, e.g., forest and ocean, cellular network is rarely available. Opportunistic communication-based architecture can perfectly solve such problems. End devices in opportunistic communication-based data collection schemes are usually low-cost sensors equipped with short-range communication module, e.g., Bluetooth. Opportunistic communication-based architecture is usually used in delay-tolerant applications [455].

In smart city, there are a large number of vehicles moving everyday, which could work as sinks to collect [70]–[72]. Bonola *et al.* propose to use vehicles to collect data through opportunistic communications [73]. In this scenario, sensors are deployed near the road. When vehicles pass these nodes, data is gathered through opportunistic communication. However, this scheme can only collect data through sensors along the road. If the sensor is far away from the road, the data cannot be gathered. Based on [73], Luo *et al.* propose a data collection scheme through moving vehicles in city, which enables multi-hop transmission among sensors [74]. Sensor nodes are clustered into several clusters. Within a cluster, all sensor nodes first send their data to the head of the cluster via multi-hop delivery. Then the gathered data is transmitted to the passing vehicle. The data collection scheme is shown as Fig.9. The cluster head is selected based on the distance with

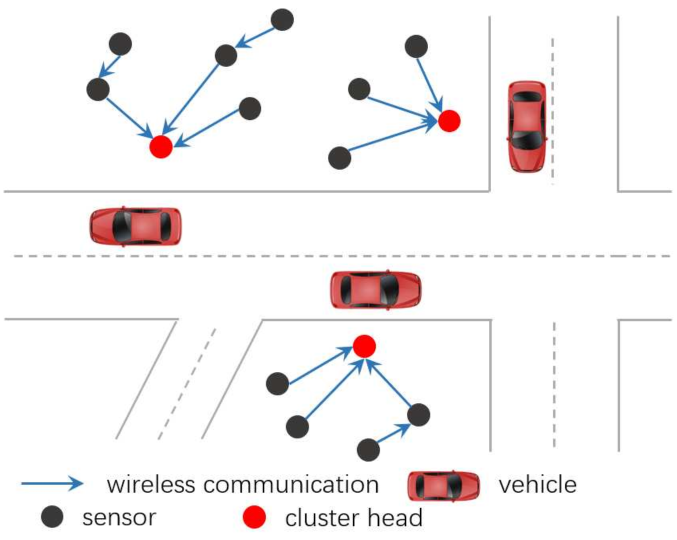


Fig. 9. Illustration of cluster-based data collection. The generated data is first sent to the cluster head, and then transmitted to the passing vehicle.

the vehicle, and these sensor nodes are clustered based on the distance with the cluster head. The work in [75] improves the cluster head selection based on road segmenting to support the collection of delay-sensitive data.

The mobility of vehicles is highly dynamic in the city, which brings a big challenge for data transmission and delivery [76], [77]. Some literature [78]–[81] focuses on data collection based on vehicle mobility prediction. Lin *et al.* study the long-term movement pattern of vehicles in the city and use K-means algorithm to estimate the contact duration between vehicles and road side units (RSUs) and the contact duration between vehicles based on their history contact records [81]. Then the data carrier vehicle greedily selects the vehicle or RSU for next hop to delivery data. The work [79] considers the scenario of recruiting several vehicles to collect data in the city. They estimate the sensing capacity of vehicles based on their future temporal distribution and Markov chain to predict the mobility of vehicles. Then a greedy approximation algorithm is used to select the optimal vehicles. Considering the low prediction accuracy in [81] and [79], Zhu *et al.* propose to use deep learning approach to accurately predict the mobility of vehicles in the recruiting problem [80]. He *et al.* further consider the budget issues in recruiting vehicles based on predicted trajectory [78]. The minimum budget problem is NP-complete and they proposed two greedy-based algorithms to solve it. Instead of predicting the mobility of vehicles, Huang *et al.* propose to recruit vehicles with fixed trajectories and runtime, e.g., bus to collect data from other sensing vehicles in the city [82]. Considering the coverage and overlapping of different buses, they propose a simulate anneal-based algorithm to solve the problem of optimal set of buses.

Smart ocean [456] is also attracting more and more attention from both academia and industry, benefiting from the fast development of underwater IoT. Large amount of data is collected by IoT devices deployed in the ocean to power applications like ocean environment monitoring, resource detection, and navigation. There are two commonly adopted architectures for underwater data collection: AUV-driven data collection [87] and UAV-driven data collection [88]. The former one usually uses AUV to directly collect data from underwater IoT devices, while the latter one usually uses relays, e.g., floating nodes in the ocean to gather data, and then forward the gathered data to UAV. Then, the AUV/UAV transmits the collected data to data center through satellite network.

2) Data management: Data management refers to the process of storing and managing the data collected from end devices to provide transparent data accessing service for edge intelligent applications. Traditional cloud-based intelligent applications require to upload data from edge to the central data center, which is easy for management and maintenance. However, in edge intelligence, data is distributively kept at the edge of network. Efficiently managing such massive amount of data is essential for performing intelligent applications [457]. There are usually two kinds of placement of edge data: edge repository and edge device.

Psaras *et al.* make a preliminary exploration on data storage and management at edge and propose to implement data repositories at base stations or access points [92]. The data repository service for users, similar with voice and data service, is also provided by the network operator. If the user moves to other places, the repository is allowed to move with the user and eventually get processed by intelligent applications. Based on [92], Nicolaescu *et al.* propose a feedback algorithm to evaluate the service quality and resource allocation of edge data repositories in different scenarios [93]. Liu *et al.* propose to use embedded storage device as data repository and study the impact of storage space and the number of data repository on data availability [94]. They find that the embedded storage device-based data repository system has much lower risk of losing data.

The work in [92] assumes that the repository moves along with users, which is not feasible sometime. Edge devices may request for computing from edge server. Since the computing capability of edge server is limited, the request may not be processed by the nearest edge server. The request may be processed by other edge server or multiple edge servers, which requires to replace the data to corresponding repositories. Hence, the optimal data block placement is of utmost importance to the success of computing request. Baktir *et al.* propose to utilize Software-Defined Networking (SDN) to orchestrate the data placement [95]. SDN can dynamically forward the data to corresponding edge data repository from the centralized view. Nicolaescu *et al.* propose to place data according to multiple criteria, e.g., data size, freshness, popularity, etc., and design a store edge networked data (SEND) system to intelligently place different types of data at edge data repositories where it is expected to be required [96]. The system is shown as Fig. 10. End devices first upload their generated data to closest edge data repository. Each repository is associated with an edge server and periodically send retrieval information to the controller. The central controller is based on SDN, which keeps monitoring the network flow of each edge data repository and predict where the data will be needed. Then the data is relocated among repositories to provide a better data service for intelligent applications. Xie *et al.* further consider

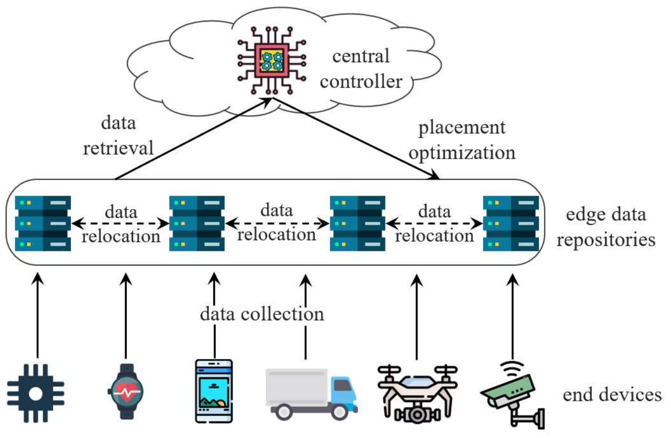


Fig. 10. Illustration of SEND system.End devices upload their data to edge data repositories. Repositories periodically report their retrieval information to the central controller and relocate data according to the data requirement prediction.

the load balance between edge data repositories and propose a greedy-based algorithm to relocate data [97].

The cached data is to be finally consumed by intelligent applications. Hence the placement has significant impact on the execution of applications, e.g., response time. Li *et al.* formulate the joint optimization of data placement and task scheduling as a 0-1 knapsack problem and propose a tabu search-based algorithm to solve it [98]. Du *et al.* further consider the scenario of sharing data among multiple users [99]. The authors model the data placement problem with multiple factors impacting the execution of applications, e.g., the storage space of repositories, bandwidth between repositories, data popularity, etc., and identify two major factors causing the delay. Then they propose a discrete particle swarm optimization (DPSO) algorithm to optimize data placement.

Edge server is usually associated with edge data repository deployed on base station, which jointly provides computing and storage service for edge devices. Moreover, benefiting from the cellular network, the connection between edge server and edge device is stable and the response latency is ultra-low. However, the cost, including the hardware cost of infrastructures and the operating cost, is very high. In contrast, the cost of storing and managing data on edge devices is negligible. The main problem of placing data on edge device is the poor storage capability of edge devices. The data with privacy concerns could be stored locally without any management cost. For example, users store their own private data on their smartphones. The shared data, e.g., road traffic, could be stored on multiple edge devices with a collaborative manner. Considering the fact that intelligent applications are driven by large amount of data, these edge devices could adopt collaborative training approaches in intelligent applications, which will be introduced in detail in Section V-A2.

Storing user's data on their own local devices is very easy to be realized. Hence, we mainly focus on the storing and management problem on multiple edge devices with collaborative manner. Relative works mainly focus on micro cloud-based schemes, especially vehicular micro cloud [100]–[103]. There are two vehicular micro cloud architectures: static

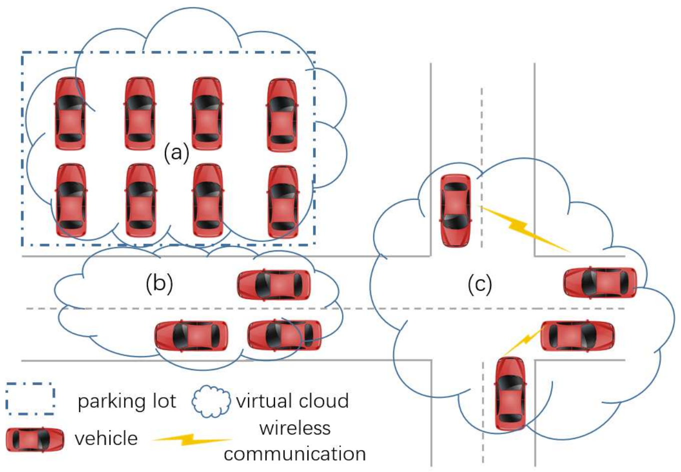


Fig. 11. Illustration of vehicular micro cloud architecture. (a) Parked vehicles form a virtual vehicular micro cloud, in which members are relatively fixed. (b) Homodromous vehicles form a vehicular micro cloud. (c) Moving vehicles form a virtual vehicular micro cloud at the intersection, in which the membership is frequently changed.

and dynamic vehicular micro cloud. The former one adopts parked vehicles to form a virtual micro cloud for data storage or computation, while the latter one maintains a virtual micro cloud with moving vehicles. As shown in Fig. 11, (a) is a static vehicular micro cloud, while (b) and (c) are dynamic vehicular micro cloud.

The key idea of vehicular micro cloud is to let vehicles connect with each other to form a virtual cloud, in which vehicles work collaboratively to provide data storage, processing, and sensing service for both members and other users. Researchers from Toyota have verified the feasibility of vehicular micro cloud through practical implementation [104] and analysis on vehicle probe datasets [105]. The shortcoming of the dynamic vehicular micro cloud is obvious. The network is unstable and easy to be decomposed due to the high-speed mobility of vehicles. Hagenauer *et al.* propose a comprehensive framework of vehicular micro cloud for parked vehicles [106]. This framework includes gateway selection, handover mechanism, and in-out management. The system dynamically selects a subset of vehicles in the parking lot as gateways to maintain the cached data and computing resource. When vehicles or users pass by, gateways seamlessly provide service for them with handover mechanism. If vehicles enter or leave the parking lot, they will inform all other gateways for data transferring. Dressler *et al.* propose a distributed hash table-based method to efficiently manage and access the data within the parked vehicular micro cloud [107], [108].

Ciocan *et al.* evaluate the performance of vehicular micro cloud in urban in terms of data preserving and find that only when the density of vehicles is high enough, performance of the cloud can meet the requirement of data preserving [109]. To deal with the problem of failure in micro cloud caused by resource scarcity, Higuchi *et al.* design a mechanism to schedule when and where to form micro cloud [110]. They assume that the probability of successful micro cloud is positively relevant with the amount of resource. Each vehicle periodically registers at RSUs to report its current resource and data. For a given region, if the amount of resource reaches the threshold, the micro cloud is formed.

In a vehicular micro cloud, data is stored at members of the cloud. Due to the mobility of vehicles, keeping the data available in a vehicular micro cloud is challenging. If there is only one copy of a data item, and the vehicle holding the data item leaves the current cloud, the data item will be lost. Pannu *et al.* propose a data management protocol to solve such problem, which enables the communication between micro clouds [111]. Specifically, when a vehicle is leaving the current cloud, it checks the redundancy of the data in its knowledge base within the cloud. If it is only copy, it broadcasts the information of the data and its moving intention. Vehicles coming to join the cloud, also interested in the data, will retrieve the data to keep the data alive.

3) Applications: Benefiting from the development of IoT and big data, smart farming is becoming more and more popular. Various kinds of intelligent algorithms are applied to improve the quality of farming. Data caching systems collect various kinds of data from environment and livestock, e.g., temperature, humidity, luminance, gyroscope, acceleration, GPS, etc., to provide massive amount of data to drive such intelligent algorithms [112]. Bhargava *et al.* develop a data caching system for smart farming [113]. In particular, two kinds of sensors are used in the scheme: in-field sensor to collect the environment status, e.g., weather, grass growth, and collar sensor on cows to monitor the health status and trajectory. When cows come back to the milking station, data is transmitted to the gateway for caching. Such data could be further used for the training of a wide range of intelligent algorithms, for example, cow behavior detection [458], [459], crop maturation time prediction [460], crop yield prediction [461], and drought detection [462]. The farming system keeps working, and the cached environment data is periodically updated, which will be further used for the inference of these algorithms to facilitate the farm management.

In transportation system, road accident detection in real time is necessary, which determines whether medical care could be provided in time. Training such an algorithm is difficult, due to the lack of accurate accident dataset [463]. To solve the problem, Khaliq *et al.* design an accident data collection system and further train an accident detection algorithm based on vehicular network [114]. Each vehicle is equipped with multiple kinds of sensors e.g., accelerometer, gyroscope, camera etc. The camera can capture the critical moment accidents. Vehicles keep sending sensed data to RSUs for caching, which is used to train accident detection algorithm. After the detection algorithm is trained, RSUs keeps monitoring the road by performing the detection algorithm. Once accident occurs, it will be detected in real time.

B. Computation caching

In the wave of AI, we are now surrounded by various intelligent edge devices, such as smartphones, smart watches, and smart bands. These intelligent edge devices provide diverse applications to augment users' understanding of their surroundings [464], [465]. For example, speech-recognition

TABLE II
LITERATURE SUMMARY OF DATA CACHING.

Ref.	Placement	Scenario	Communication	Content	Strategy	Metric
[51]	Data center	General	Cellular	Data Collection	Direct collection	Model accuracy
[52]	Data center	Indoor navigation	Cellular	Data Collection	Direct collection	Model accuracy
[53]	Data center	Indoor sensing	Cellular	Data Collection	Direct collection	Model accuracy
[54]	UAV	Unfriendly environment	Opportunistic	Data collection	Path optimization	Data collection time
[55]	IoT device	Wireless sensor network	Opportunistic	Trust evaluation	Trust mechanism	Energy consumption
[56]	IoT device	Wireless sensor network	Opportunistic	Trust evaluation	Digital signature	Cost of evaluation
[57]	IoT device	Wireless sensor network	Opportunistic	Data fairness	Flying speed control	Fairness
[73]	Vehicle	Roadside data collection	Opportunistic	Data collection	Oblivious data mules	Experimental evaluation
[74]	Vehicle	Smart city	Opportunistic	Data collection	Clustering	Energy consumption
[75]	Vehicle	Delay-sensitive	Opportunistic	Data collection	Road segmenting	Supported data types
[81]	Vehicle	Mobile crowdsensing	Opportunistic	Data collection	Mobility prediction	Delivery rate
[79]	Vehicle	Mobile crowdsensing	Opportunistic	Data collection	Mobility prediction	Data collection efficiency
[80]	Vehicle	Mobile crowdsensing	Opportunistic	Data collection	Mobility prediction	Collected data volume
[78]	Vehicle	Mobile crowdsensing	Opportunistic	Data collection	Trajectory prediction	Coverage
[82]	Bus	Mobile crowdsensing	Opportunistic	Data collection	Collector selection	Redundancy rate
[86]	Sensor	Wireless sensor network	Opportunistic	Data aggregation	Early message mechanism	Aggregation probability
[87]	IoT device	Underwater sensing	Opportunistic	Path connectivity	Sink-node-deployment	Fairness
[88]	AUV	Underwater sensing	Opportunistic	Data collection	Shortest path tree	Energy consumption
[92]	Data repository	General	Cellular	Data management	Framework design	Hypothesis
[93]	Data repository	General	Cellular	Data management	Feedback evaluation	Satisfaction rate
[94]	Data repository	Embedded storage	Cellular	Data placement	System design	Data loss rate
[95]	Data repository	SDN	Cellular	Data placement	SDN orchestration	Service delay
[96]	Data repository	SDN	Cellular	Data placement	Requirement prediction	Retrieval time
[97]	Data repository	SDN	Cellular	Data placement	Load balance	Routing path length
[98]	Edge server	General	Cellular	Data placement	Tabu search	Data response time
[99]	Edge server	Data sharing	Cellular	Data placement	DPSO	Data transmission time
[104]	Vehicular cloud	Stop interaction	Opportunistic	Data management	Micro cloud demo	Demo
[105]	Vehicular cloud	Interaction, freeway	Opportunistic	Data management	Probe datasets analysis	Feasibility
[106]	Vehicular cloud	Parking lot	Opportunistic	Data management	Micro cloud framework	Data transmitting rate
[107]	Vehicular cloud	Parking lot	Opportunistic	Data management	Distributed hash table	N/A
[109]	Vehicular cloud	Urban environment	Opportunistic	Data management	Real trace experiments	Replicas number
[110]	Vehicular cloud	General	Opportunistic	Data management	Cloud formation	Failure frequency
[111]	Vehicular cloud	Intersection	Opportunistic	Data management	Data recovery	Data availability

based AI assistants, e.g., Siri and Cortana, and song identification enabled music applications, have been widely used in people's daily lives. Such AI-based applications are of high computational complexity and cause high power consumption of the device [396], [397], [466]. Meanwhile, some researchers have discovered the computation redundancy in AI-based applications, e.g., face detection and voice recognition, which indicates the feasibility of computation reuse for AI applications [115]. It saves ridiculous amount of time and energy for edge devices performing AI applications through caching the results of previous computation [116].

The key idea of computation caching technologies is to explore the spatiotemporal redundancy of computation. Computation redundancy is caused by commonly used high computational complexity applications or AI models. Generally, there are three kinds of computation caching, according to the granularity of computation redundancy: layer-level, model-level, and device-level computation caching. Table III summaries existing literature on computation caching.

1) Layer-level computation caching: CNN and its variations achieve state-of-the-art performance in computer vision. In CNN-driven applications, input images are processed through CNN layer by layer to produce intermediate results, which is also called feature maps, and then output the inference results. On the other hand, in continuous mobile vision, there is plenty of redundant information between continuous frames. Hence, the intermediate results are reusable to avoid redundant computation for users. Such paradigm is also called incremental inference [467]

The key question is what should be cached. In continuous vision, the content of frames varies along with time, which means directly caching the computation result of a complete frame is not possible. Xu *et al.* design a layer-level caching framework, called DeepCache, for continuous mobile vision applications, in which each frame is split into multiple regions of same size (10x10 pixels) [117]. There are large number of redundant regions among different frames. As shown in Fig. 12, the region is treated as cache key, while the result, i.e., feature map of the region is cache value. A lookup table is created to map regions to feature maps. When forward propagation is performed at each layer, the cached feature maps are used to replace the actual execution. Experiments on image classification with ncnn framework show that the proposed caching system could speed up the inference significantly with less than 3% accuracy loss. However, comparing and identifying these reusable regions from each frame requires significant overhead, which limits the accelerating performance and introduce extra energy consumption. Moreover, due to the errors in region matching with cache keys, the accuracy of the model would be also degraded.

The focused scenario of [117] is general. Huynh *et al.* design a simplified layer-level caching system for mobile continuous vision of first-person-view [118]. In continuous first-person-view based applications, there are not too much

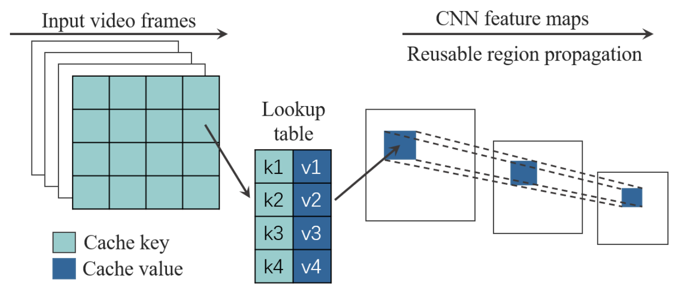


Fig. 12. Illustration of DeepCache. Each frame is partitioned into multiple regions. The lookup table maps the cache keys, i.e., regions to cache value, i.e., feature maps.

changes between consecutive images. Hence, the authors propose to directly compare regions in the same location in consecutive images to identify reusable feature maps, which could significantly reduce the computation overhead compared with [117]. However, comparing two images is computation consuming, even for 8x8 grids. To solve this problem, the authors propose to use the color distribution to compute the similarity of two images. Specifically, they calculate the chi-square distance of two color distributions. Two images are treated as similar if the distance is smaller than the threshold. Although the proposed color distribution-based approach reduces computation load, it results in further accuracy loss. Cavigelli *et al.* propose to not only compare regions of successive images at the input layer but also before every intermediate layer to reduce accuracy loss in similar application scenario [119], [120]. Similar approach is also adopted in [121]. Inspired by video codecs [122], Buckler *et al.* propose an activation motion compensation (AMC) algorithm to detect changed pixels and incrementally update the output for predicted frames [123]. Different from [117], [118], AMC requires to manually split the CNN model into two parts. The result of prefix part is cached. AMC only need to execute the suffix part to compensate for the changed pixels.

In some mobile continuous computer vision applications, such as driving assistance, the system is required to provide high trackability. The system needs to recognise, locate, and label the tracked object, e.g., road signs, on the screen in real-time, which makes it hard to execute the recognition model on a resource-constrained edge device. Chen *et al.* develop an active cache based continuous object recognition system, called Glimpse, with the assistance of cloud server to achieve real-time trackability [124]. The structure of Glimpse is shown as Fig. 13. Glimpse caches frames locally and only uploads trigger frames to the cloud server. Trigger frames refers to the frames, for which the recognition from the server is different from current local tracking. The cloud server sends back the recognised object, its labels, bounding boxes, and features, which would be cached locally on devices. Then the devices would track the object with the labels, bounding boxes, and features locally on captured frames. A similar approach is also adopted in CNNCache [46]. However, detecting these trigger frames also requires intensive computation. Naderiparizi *et al.* develop a framework, Glimpse, to select valid frames by performing coarse visual processing with low energy con-

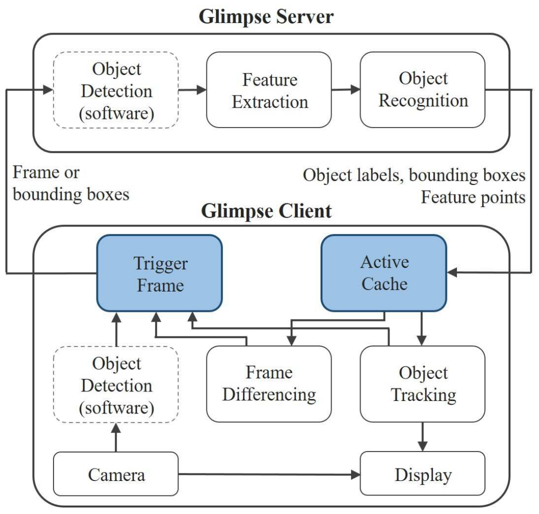


Fig. 13. The architecture of Glimpse. Edge device, i.e., glimpse client, only uploads trigger frames to the cloud to save bandwidth resources. Glimpse server transmits the recognition results and features back to edge devices. Edge devices deal with local frames with these features.

sumption [125]. Glimpse adopts gating sensors and redesigns the processing pipeline to save energy.

2) Application-level computation caching: Modern mobile systems have the advantage of good compatibility, support various vision applications [468]. These vision application on the same mobile device share the same root data, as well as the processing components, which provides the possibility of sharing the computation result across applications.

Likamwa *et al.* propose a across-application caching framework, named Starfish, which enables computation sharing among concurrent computer vision applications on the same device [126]. As the name suggests, there is a core, i.e., a vision library in the center of the framework, which processes root data and caches the result. When vision applications call for the library, the core returns the cached results, which significantly reduce the computation overhead. However, this framework only support applications of the same type, which indicates poor scalability. To solve this problem, Guo *et al.* design Potluck, an application-level caching system for approximate caching [127]. They investigate the temporal, spatial and semantic correlation of input of different applications. When an application requests for computation, the input is processed into a feature vector with fixed length, which is treated as the cache key. If the key is similar to an existing key within a given threshold, the cache hits. Otherwise, the input is processed, and the results will be cached.

There are also some research focusing on static scenarios, where the captured images are all predictable. Boos *et al.* design FlashBack, a near real-time immersive virtual reality (VR) system based on pre-caching all possible images [128]. Before performing VR application, FlashBack renders all possible images or download pre-rendered results from the server and cache it on local device. When performing VR application, a CacheManager component fetches rendered results according to user's location and orientation. Obviously, human-interaction is not supported by the system. The other disadvantage is the giant storage load. It takes about 50GB for a single application, which is unacceptable for resource-constraint edge devices. Qian *et al.* solve the problem of overwhelming storage load through pre-caching the results of future viewport, instead of the complete pre-rendered results from the server based on the prediction of head movement [129]. They study the head movement behaivor of 130 diverse users when they are watching immersive video and use four ML algorithms to predict users head movement. Based on the movement prediction of next time slot, the system pre-caches the corresponding results from the server.

3) Device-level computation caching: Since most edge devices are resource-constraint, offloading computation-intensive tasks to edge server is a promising solution. From the perspective of users of the same area, the requested recognition tasks would exhibit spatio-temporal locality. For example, in a park, nearby visitors may use their smartphones to recognise the same flowers and then search the information accordingly. In this case, these recognition tasks are offloaded to the same edge server. If the results of previous recognition tasks could be used for latter tasks, the response time would be significantly decreased. In [47], Guo *et al.* crawl street views by using the Google Streetview API [48] and builds an 'outdoor input image set'. They then find that around 83% of the images exhibit redundancy, which leads to a large number of potential unnecessary computations for image re-organization application. Also, they analyse NAVVIS [49], an indoor view dataset, and observed that nearly 64% of indoor images exhibit redundancy.

Device-level computation caching usually adopts server-client architecture. Drolia *et al.* propose a caching framework, Cachier, to cache the recognition results on edge servers [130]. Specifically, Cachier first extracts features of the requested recognition task, and then tries to match a similar object from the cache. If there is a cache hit, the corresponding computation results would be sent back to the mobile device. Otherwise, the request would be sent to the cloud. To identify similar recognition tasks, they used a Locality Sensitive Hashing (LSH) algorithm [131] to determine the best match. Furthermore, to overcome the unbalanced and time-varying distribution of users' requested tasks, Guo *et al.* design an Adaptive LSH-Homogenized kNN joint algorithm which outperforms LSH in terms of evaluation results [47]. Drolia *et al.* further introduce a proactive caching strategy into their system by predicting the requirement of users and proactively caching parts of models on edge server for pre-processing to further reduce the latency [132]. Such a strategy is also used in [133] to deal with unstructured data at edge.

Moreover, in some task-fickle scenarios, multiple different kinds of tasks, e.g., voice recognition and object recognition, are offloaded from devices to edge server. By pre-caching multiple kinds of deep learning models on server for different kinds of tasks, we can reduce the computation time and further improve users' QoE. Taylor *et al.* propose an adaptive model selection scheme to select the best model for users [134]. They use a supervised learning method to train a predictor model offline and then deploy it on an edge server. When a request arrives, the predictor will select an optimal model for the task. In [135], Zhao *et al.* propose a system, Zoo, to compose different models to provide a satisfactory service for users. Ogden *et al.* propose a deep inference platform, MODI, to determine what model to cache and what model to use for specific tasks [136]. There is a decision engine inside MODI, which aggregates previous results to decide what new models are required to cache. Aforementioned solutions require users to upload data to edge server for processing, which leads to relatively high latency (but much lower than cloud-based solutions). Fang *et al.* propose a caching scheme with the joint consideration of latency and accuracy [137]. A complex model is partitioned and distributed between edge devices and the cloud server. The input data is first processed locally and then sent to edge server for further process.

4) application: Tracking eye gaze is widely used in various scenarios, for example hand-free interaction in games and irregular behavior detection in driving. However, tracking eye in real-time is computation-intensive, which needs to process eye images with a quite high frame rate. Mayberry *et al.* design a computation caching enabled mobile gaze tracker on wearable device [138]. There is large amount redundant information on eye images. Hence, processing results on these repeated regions are cached. For each eye image, they authors propose a point-of-gaze predictor model to estimate the location of eye on eye image. They only need to execute inference on a small region containing eye.

In face recognition, only some landmarks, e.g., eye, mouth, nose, etc., are useful. Most pixels are occupied by the background, skin and hair in human face images, which are of high redundancy. Kim *et al.* develop a similar feature skipping module for face recognition CNN algorithm [469]. They use L1 distance quantify the similarity of input features. Convolution computation of repeated features are skipped, which saves large amount of computing operations and speed up the recognition inference.

Current voice assistants are based on cloud computing, which introduce extra latency. Xu *et al.* apply caching mechanism on home voice assistant system to reduce the interaction with cloud server [139]. They find that the usage of voice assistant at home follows a power-law distribution. Although people's interest is diverse, commonly used command for voice assistant could be covered by three domains. Hence, computation results of these commonly used commands are cached locally. Only when cache misses, cloud service is used to process the new command. Similarly, caching mechanism also applies to instance recognition. Lovagnini *et al.* cache redundant instance recognition on cloud environment to reduce the latency for users [140].

V. Edge Training

The standard learning approach requires centralising training data on one machine, whilst edge training relies on distributed training data on edge devices and edge servers, which is more secure and robust for data processing. The main idea of edge training is to perform learning tasks where the data is generated or collected with edge computing resources.

TABLE III
LITERATURE SUMMARY OF COMPUTATION CACHING.

Ref.	Granularity	Scenario	Enabler	Baseline (without caching)	Performance
[116]	Application	Image recognition	compute reuse	Face-detection [470]	50× faster
[117]	Layer	Mobile vision	Results lookup	AlexNet [471], GoogLeNet [267] YOLO [331] ResNet-50 [268]	20% faster
[118]	Layer	Mobile vision	Grid comparison	VGG-Verydeep-16	From 3s to 644ms
[119]	Layer	Computer vision	Input decomposition	cuDNN baseline	9.1× faster
[121]	Layer	Computer vision	Input decomposition	VGG 19 [472]	10× faster
[123]	Layer	Computer vision	Motion compensation	AlexNet, Faster16, FasterM	87% energy reduced
[124]	Layer	Mobile vision	Trigger detection	GoogleNet [267]	improve battery life by 24%
[125]	Layer	Mobile vision	Frame early-discarding	RGB tracking algorithm [473]	7-25× less energy
[126]	Application	Computer vision	Library splitting	Face, Object Recognition Scene Geometry	19%-58% less energy
[127]	Application	Computer vision	Approximate deduplication	AlexNet [471]	Lower latency
[128]	Application	Virtual reality	Rendering Memoization	Viking Village	97× less energy
[129]	Application	Virtual reality	Viewport prediction	Festive [474], BBA [475] Full [476], H2 [477]	18× quality improvement
[47]	Device	Computer vision	Scalable lookup	Google Lens	Lower latency and energy cost
[130]	Device	Image recognition	System design	Offloading to cloud	3× speedup
[132]	Device	Image recognition	Requirement prediction	Markov model	5× speedup
[134]	Device	Image recognition	Model selection	Inception [478], ResNet [479] MobileNet [231]	1.8× faster
[135]	Device	Recommender system	Composable services	Inception [478], LeNet-5 [480] VGG16 [472]	Lower latency
[136]	Device	General	Model selection	Inception V3 [481]	2.3× faster
[137]	Device	General	Model Partitioning	Complete model	58-217% faster

It is not necessary to send users' personal data to a central server, which effectively solves the privacy problem and saves network bandwidth.

Training data could be solved through edge caching. We discuss how to train an AI model in edge environments in this section. Since the computing capacity on edge devices and edge servers is not as powerful as central servers, the training style changes correspondingly in the edge environment. The major change is the distributed training architecture, which must take the data allocation, computing capability, and network into full consideration. New challenges and problems, e.g., training efficiency, communication efficiency, privacy and security issues, and uncertainty estimates, come along with the new architecture. Next, we discuss these problems in more detail.

A. Training architecture

Training architecture depends on the computing capacity of edge devices and edge servers. If one edge device/server is powerful enough, it could adopt the same training architecture as a centralised server, i.e., training on a single device. Otherwise, cooperation with other devices is necessary. Hence, there are two kinds of training architectures: solo training, i.e., perform training tasks on a single edge device/server, and collaborative training, i.e., few devices and servers work collaboratively to perform training tasks.

1) Solo training: Early researchers mainly focus on verifying the feasibility of directly training deep learning models on mobile platforms. Chen *et al.* find that the size of neural network and the memory resource are two key factors that affect training efficiency [141]. For a specific device, training efficiency could be improved significantly by optimising the model. Subsequently, Lane *et al.* successfully implement a constrained deep learning model on smartphones for activity recognition and audio sensing [36]. The demonstration achieves a better performance than shallow models, which demonstrates that ordinary smart devices are qualified for simple deep learning models. Similar verification is also done on wearable devices [37] and embedded devices [38].

2) Collaborative training: The most common collaborative training architecture is the master-slave architecture. Federated learning [39] is a typical example, in which a server employs multiple devices and allocates training tasks for them. Li *et al.* develop a mobile object recognition framework, named DeepCham, which collaboratively trains adaptation models [142]. The DeepCham framework consists of one master, i.e., edge server and multiple workers, i.e., mobile devices. There is a training instance generation pipeline on workers that recognises objects in a particular mobile visual domain. The master trains the model using the training instance generated by workers. Huang *et al.* consider a more complex framework with additional scheduling from the cloud [143]. Workers with training instances first uploads a profile of the training instance and requests to the cloud server. Then, the cloud server appoints an available edge server to perform the model training.

Peer-to-peer is another collaborative training architecture, in which participants are equal. Valerio *et al.* adopt such training architecture for data analysis [144]. Specifically, participants first perform partial analytic tasks separately with their own data. Then, participants exchange partial models and refine them accordingly. The authors use an activity recognition model and a pattern recognition model to verify the proposed architecture and find that the trained model could achieve similar performance with the model trained by a centralised server. Similar training architecture is also used in [145] to enable knowledge transferring amongst edge devices.

B. Training Acceleration

Training a model, especially deep neural networks, is often too computationally intensive, which may result in low training efficiency on edge devices, due to their limited computing capability. Hence, some researchers focus on how to accelerate the training at edge. Table IV summaries existing literature on training acceleration.

Chen *et al.* find that the size of a neural network is an important factor that affects the training time [141]. Some efforts [146], [147] investigate transfer learning to speed up the training. In transfer learning, learned features on previous models could be used by other models, which could significantly reduce the learning time. Valery *et al.* propose to transfer features learned by the trained model to local models, which would be re-trained with the local training instances [146]. Meanwhile, they exploit the shared memory of the edge devices to enable the collaboration between CPU and GPU. This approach could reduce the required memory and increase computing capacity. Subsequently, the authors further accelerate the training procedure by compressing the model by replacing float-point with 8-bit fixed point [147].

In some specific scenarios, interactive machine learning (iML) [482], [483] could accelerate the training. iML engages users in generating classifiers. Users iteratively supply information to the learning system and observe the output to improve the subsequent iterations. Hence, model updates are more fast and focused. For example, Amazon often asks users targeted questions about their preferences for products. Their preferences are promptly incorporated into a learning system for recommendation services. Some efforts [148], [150] adopt such approach in model training on edge devices. Shahmohammadi *et al.* apply iML on human activity recognition, and find that only few training instances are enough to achieve a satisfactory recognition accuracy [150]. Based on such theory, Flutura *et al.* develop DrinkWatch to recognise drink activities based on sensors on smartwatch [151].

In a collaborative training paradigm, edge devices are enabled to learn from each other to increase learning efficiency. Xing *et al.* propose a framework, called RecycleML, which uses cross modal transfer to speed up the training of neural networks on mobile platforms across different sensing modalities in the scenario that the labelled data is insufficient [145]. They design an hourglass model for knowledge transfer for multiple edge devices, as shown in Fig. 14. The bottom part denotes lower layers of multiple specific models, e.g., AudioNet, IMUNet, and VideoNet. The middle part represents the common layers of these specific models. These models project their data into the common layer for knowledge transfer. The upper part represents the task-specific layers of different models, which are trained in a targeted fashion. Experiments show that the framework achieves 50x speedup for the training. Federated learning could be also applied to accelerate the training of models on distributed edge devices. Smith *et al.* propose a systems-aware framework to optimise the setting of federated learning (e.g., update cost and stragglers) and to speed up the training [152].

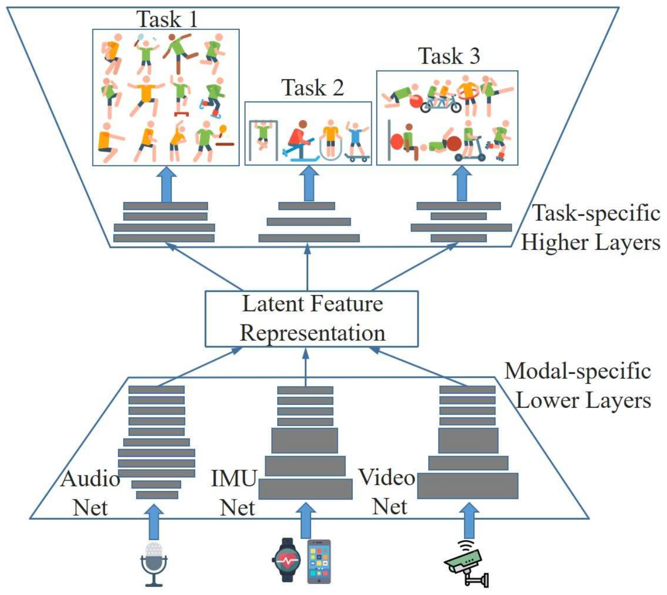

Task 1
Task 2
Task 3
Task-specific Higher Layers
Latent Feature Representation
Modal-specific Lower Layers
Audio Net
IMU Net
Video Net

Fig. 14. Illustration of the hourglass model. The lower part represents lower layers of specific sensing models. The latent feature representation part is the common layer. Lower layers project their data into this layer for knowledge transfer. The upper part represents task-specific higher layers, which are trained for specific recognition tasks.

C. Training optimization

Since solo training is similar to training on a centralised server to a large extent, existing work mainly focuses on collaborative training. Federated learning is the most typical collaborative training architecture, and almost all literature on collaborative training is relevant to this topic. Hence, we focus on the optimization on federated learning here.

Federated learning is a kind of distributed learning [144], [484], [485], which allows training sets and models to be located in different, non-centralised positions, and learning can occur independent of time and places. This learning paradigm is first proposed by Google, which allows smartphones to collaboratively learn a shared model with their local training data, instead of uploading all data to a central cloud server [39].

The learning process of federated learning is shown as Fig. 15(a). There is a untrained shared model On the central server, which will be allocated training participants for training. Training participants, i.e., edge devices train the model with the local data. After local learning, changes of the model are summarised as a small focused update, which will be sent to the central server through encrypted communication. The central server averages received changes from all mobile devices and updates the shared model with the averaged result. Then, mobile devices download the update for their local model and repeats the procedure to continuously improve the shared model. In this learning procedure, only the changes are uploaded to the cloud and the training data of each mobile user remains on mobile devices. Transfer learning and edge computing are combined to learn a smarter model for mobile users. In addition, since learning occurs locally, federated learning could effectively protect user privacy, when compared

TABLE IV
LITERATURE SUMMARY OF MODEL ACCELERATION IN TRAINING.

Ref.	Model	Enabler	Learning method	Scenario	Performance
[141]	DNN	Hardware acceleration	Transfer learning	Mobile computing	Prototype of TensorFlow+
[146]	CNN	Hardware acceleration	Transfer learning	Mobile computing	Faster than Caffe-OpenCL trained
[147]	CNN	Hardware acceleration parameter quantisation	Transfer learning	Mobile computing	Faster than Caffe-OpenCL trained
[149]	DNN	Hardware acceleration	Transfer learning	Analog computing	Close to software baseline of 97.9
[150]	RF, ET, NB LR, SVM	Human annotation	Incremental learning	Human activity recognition	93.3% accuracy
[151]	Naive Bayes	Human annotation	Incremental learning	Human activity recognition	6-8 hours to train a model
[145]	CNN	Software acceleration	Transfer learning	Human activity recognition	50× faster than scratch training
[152]	Statistical model	Software acceleration	Federated learning	Multi-task learning	Outperform global, local manners
[153]	GCN	Software-hardware	Graph learning	Mobile computing	15× faster than cpu-only scheme

with a centralised learning approach.

The aforementioned federated learning is cloud-based. Besides, there are two other kinds of federated learning: edge-based federated learning, and hierarchical federated learning. The architecture comparison of these three kinds of federated learning is shown as Fig. 15. In edge-based federated learning, edge servers could replace the role of cloud servers to iteratively aggregate updates from edge devices, as shown in Fig. 15(b). This is a two-layer architecture. Edge server recruits participants for collaborative training. The hierarchical federated learning is a combination of these two paradigms, as shown in Fig. 15(c). In each training round, edge servers hire edge devices within their coverage to train AI models. After receiving updates from edge devices, edge servers upload their aggregated updates to the central cloud server for further aggregation. Then the central server allocates the updated parameters to each edge server, which will be further allocated to edge devices.

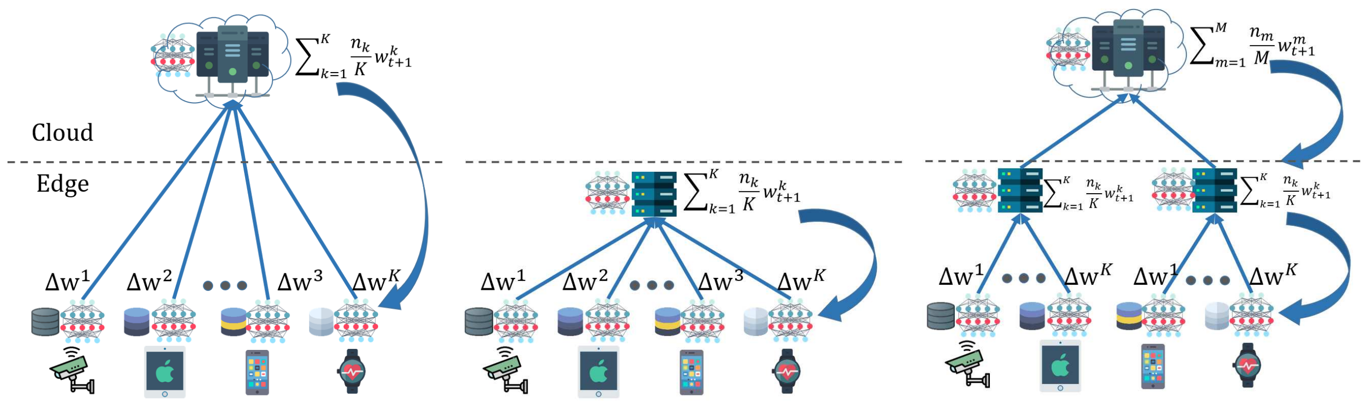
The key challenge for cloud-based federated learning is the Unstable network connection and low bandwidth. Typical edge devices are smartphones with unreliable and slow network connections. Moreover, due to the unknown mobility, these devices may be intermittently available for working. Hence, the communication efficiency between smartphones and the central server is of the utmost importance to the training. Specifically, there are two factors affecting the communication efficiency: communication frequency and communication cost. For edge-based federated learning, there are two main challenges. (i) Limited resources: smartphones have less powerful CPUs, limited storage and small batteries, let alone IoT devices and embedded devices. (ii) Scarce participants and data for training: different from cloud-based federated learning, which could recruit participant across the giant network, there are scarce edge devices in an edge area. Hierarchical federated learning could solve the data scarcity problem to some extent, since it could recruit more participants and use more data for training. In addition, the update from edge devices is vulnerable to malicious users. Hence privacy and security issue are also challenging for these three kinds of federated learning. We discuss these challenges and solutions in detail next. Table V summarises literature on federated learning.

1) Communication efficiency: In federated learning, communication between edge devices and the cloud server is the most important operation, which uploads the updates from edge devices to the could server and downloads the aggregated update from the shared model to local models. Due to the possible unreliable network condition of edge devices, minimising the number of update rounds, i.e., communication frequency between edge devices and cloud server is necessary. Jakub *et al.* are the first to deploy federated learning framework and propose the setting for federated optimisation [154]. In [155], the authors characterise the training data as massively distributed (data points are stored across massive edge devices), non-IID (training set on devices may be drawn from different distributions), and unbalanced (different devices have different number of training samples). In each round, each device sends an encrypted update to the central server. Then they propose a federated stochastic variance reduced gradient (FSVRG) algorithm to optimise the federated learning. They find that the central shared model could be trained with a small number of communication rounds.

McMahan *et al.* propose a federated averaging algorithm (FedAvg) to optimise federated learning in the same scenario in [154], [155] and further evaluate the framework with five models and four datasets to proof the robustness of the framework [156]. Although FedAvg could achieve good performance on certain datasets, Zhao *et al.* find that using this algorithm to train CNN models with highly skewed non-IID dataset would result in the significant reduction of the accuracy [157]. They find the accuracy reduction results from the weight divergence, which refers to the difference of learned weights between two training processes with the same weight initialisation. Earth mover's distance (EMD) between the distribution over classes on each mobile device and the distribution of population are used to quantify the weight divergence. They then propose to extract a subset of data, which is shared by all edge devices to increase the accuracy.

Strategies that reduce the number of updates should be on the premise of not compromising the accuracy of the shared model. Wang *et al.* propose a control algorithm to determine the optimal number of global aggregations to maximise the efficiency of local resources [159]. They first analyse the convergence bound of SGD based federated learning. Then, they propose an algorithm to adjust the aggregation frequency in real-time to minimise the resource consumption on edge devices, with the joint consideration of data distribution, model characteristics, and system dynamics.

Above-mentioned works adopt a synchronous updating


$\sum_{k=1}^{K} \frac{n_k}{K} w_{t+1}^{k}$
Cloud
Edge
Δw^1
Δw^2
Δw^3
Δw^K
$\sum_{m=1}^{M} \frac{n_m}{M} w_{t+1}^{m}$

(a) Cloud-based federated learning (b) Edge-based federated learning (c) Hierarchical federated learning

Fig. 15. The architectures of federated learning. Cloud-based federated learning adopts a two-layer architecture, i.e., cloud layer and client layer. Edge-based federated learning involves edge layer and client layer, while hierarchical federated learning involves cloud layer, edge layer and client layer.

method, where in each updating round, updates from edge devices are first uploaded to the central server, and then aggregated to update the shared model. Then the central server allocates aggregated updates to each edge device. Some researchers think that it is difficult to synchronise the process. On one hand, edge devices have significantly heterogeneous computing resources, and the local model are trained asynchronously on each edge device. On the other hand, the connection between edge devices and the central server is not stable. Edge devices may be intermittently available, or response with a long latency due to the poor connection. Wang *et al.* propose an asynchronous updating algorithm, called CO-OP through introducing an age filter [160]. The shared model and downloaded model by each edge device would be labelled with ages. For each edge device, if the training is finished, it would upload its update to the central server. Only when the update is neither obsolete nor too frequent, it will be aggregated to the shared model. However, most works adopt synchronous approaches in federated learning, due to its effectiveness [156], [165].

In addition to communication frequency, communication cost is another factor that affects the communication efficiency between edge devices and the central server. Reducing the communication cost could significantly save bandwidth and improve communication efficiency. Konevcny *et al.* propose and proof that the communication cost could be lessened through structured and sketched updates [155], [166]. The structured update means learning an update from a restricted space that could be parametrised with few variables through using low rank and random mask structure, while sketched update refers to compressing the update of the full model through quantisation, random rotations and sub-sampling.

Lin *et al.* find most of the gradient update between edge devices and the central server is redundant in SGD based federated learning [167]. Compressing the gradient could solve the redundancy problem and reduce the update size. However, compression methods, such as gradient quantisation and gradient sparsification would lead to the decreased accuracy. They propose a deep gradient compression (DGC) method to avoid the loss of accuracy, which use momentum correction and local gradient clipping on top of the gradient sparsification. Hardy *et al.* also try to compress the gradient and propose a compression algorithm, called AdaComp [168]. The basic idea of AdaComp is compute staleness on each parameter and remove a large part of update conflicts.

Smith *et al.* propose to combine multi-task learning and federated learning together, which train multiple relative models simultaneously [152]. It is quite cost-effective for a single model, during the training. They develop an optimisation algorithm, named MOCHA, for federated setting, which allows personalisation through learning separate but related models for each participant via multi-task learning. They also prove the theoretical convergence of this algorithm. However, this algorithm is inapplicable for non-convex problems.

Different from the client-to-server federated learning communication in [155], [167], [168], Caldas *et al.* propose to compress the update from the perspective of server-to-client exchange and propose Federated Dropout to reduce the update size [169]. In client-to-server paradigm, edge devices download the full model from the server, while in a server-to-client paradigm, each edge device only downloads a sub-model, which is a subset of the global shared model. This approach both reduces the update size and the computation on edge devices.

2) *Resource scheduling:* Federated learning requires participants iteratively execute training process and transmit updates to the server, which is energy consuming. Battery resource is of utmost importance for edge devices, which is usually very limited. In particular, there are many factors impacting the energy consumption, e.g., CPU frequency, allocated time, bandwidth, wireless channel status, convergent accuracy, etc. Yang *et al.* formulate the resource scheduling as an energy minimization problem with joint consideration of computation and update transmission [170]. They assume the local training stops until a given accuracy is reached. Through theoretical analysis, they find the time used for training is a convex function of the accuracy. Then they propose a iterative algorithm to derive the optimal resource scheduling solution. Mo *et al.* study the joint optimization problem with two wireless transmission protocols, i.e., NOMA and TDMA [171].

Different from [170], they assume execution time for edge devices is given. The overall minimum energy consumption problem is formulated as a convex function of multi factors, i.e., transmission power, transmission rate, and CPU frequency.

These works are based on the assumption that edge devices are not heterogeneous. However, in practical, edge devices are heterogeneous in terms of computing capability, battery, data distribution, network connection, etc. In federated learning, the model update is synchronized, which means edge devices with high computing capability would finish local update in advance. These devices could execute model training with lower CPU frequency to reduce energy consumption. Based on this idea, Zhan *et al.* optimize the computing resource control problem under dynamical network condition with deep reinforcement learning [172]. Tang *et al.* investigate a similar problem in UAV-enabled IoT network with deep deterministic policy gradient (DDPG) algorithm [173]. Similarly, edge devices with low computing capability or poor channel condition would prolong the local execution time, which will further postpone the synchronization with central server. Nishio *et al.* design a protocol, named FedCS, which enable the edge server set deadline for participants to upload updates and download aggregated parameters [174]. In each round, edge server greedily selects participants according to their computing capability, battery level, and network condition to aggregate updates as much as possible. Focusing on fast convergence, Mohammed *et al.* formulate the selection of best R clients from total candidates with given budget as a secretary problem and solve it with an online heuristic solution [175].

In scenarios where participants are IoT devices, which are not able to run a complete model, offloading to edge server is a feasible solution. Ye *et al.* propose to partition a CNN model and offload the suffix part to edge server in federated learning [176]. IoT device run the prefix part, which extracts features from input. Only offloading the intermediate results from the prefix part to edge server is also privacy-preserving.

In fact, not all users are willing to participate in federated learning. Some researchers focus on rewarding schemes to attract users to participate in the training procedure, especially users with good data quality and computing capability. Kang *et al.* design a contract theory-based incentive mechanism to reward participants [177]. Specifically, they define a parameter that is related to the accuracy of training model to denote the data quality. Users with high data quality can choose contract item with higher profit. Zhan *et al.* use deep reinforcement learning to learn the optimal pricing scheme for participants [178]

3) Hierarchical federated learning: Cloud-based federated learning benefits from sufficient participants and data but faces the challenge of communication cost and long latency. Edge-based federated learning benefits from the low communication cost and high bandwidth but with scarce participants and data. Hierarchical federated learning is first proposed in [179] to combine the advantages of these two kinds of learning paradigms. Edge server orchestrate edge devices within its coverage for federated learning and uploads the primary aggregation to cloud server through backbone network. The authors prove through theoretical analysis that the FedAvg algorithm can still converge in hierarchical architecture. They also study the problem of aggregation frequency at edge level and cloud level to guarantee the fast convergence of the learning process. Abad *et al.* apply the hierarchical federated learning architecture on cellular network, where base station act like the cloud server and small base station work as edge server [180]. Meanwhile, the resource allocation is optimized among edge devices to reduce the overall communication latency.

The heterogeneity of edge devices is not considered in [179], [180]. In scenarios of large amount of heterogeneous edge devices with different computing capabilities, energy levels, and data qualities being recruited to participated in federated learning, it is necessary to consider the resource allocation to reduce the overall training cost. Luo *et al.* consider the joint optimization of computation and communication resource allocation among heterogeneous edge devices in hierarchical federated learning aiming at minimum energy consumption [181]. The authors decompose the problem to reduce the complexity and solve them with iterative cost reduction strategy.

In fact, the data distribution is not independent and identically distributed, due to the heterogeneity of users, which is big challenge for federated learning. It is not possible to directly share the data among users to reduce the distribution difference. Benefiting from the dense implementation of small base stations and user mobility, in hierarchical federated learning, users are allowed to connect to multiple edge servers, which are associated with small base stations. Mhaisen *et al.* find that upper bond of parameters in cloud-based federated learning is proportional to the distribution distance between users' dataset and the global dataset [182]. Hence, they propose to associate edge devices with the optimal edge server to minimize the overall distance, which is proved to be NP-hard. The problem could be simplified and solved with heuristic algorithms. Briggs *et al.* propose to cluster participants based on the similarity of their updates and separately train model for each cluster [183]. Because federated training on non-IID would result in overfitting on local data. Similar participants could collaboratively train a specialised model. Experiments show that such method could converge quickly with less training rounds. Similarly, Chai *et al.* propose to cluster vehicles based on regional features for knowledge sharing in federated learning and introduce blockchain to guarantee the security [184]. Each cluster maintains a blockchain to record the shared model. RSU is a given region acts as the role of edge server for primary aggregation. They also design a light-weight PoK mechanism to reduce computation overhead.

4) Privacy and security issues: After receiving updates from edge devices, the central server needs to aggregate these updates and construct an update for the shared global model. Currently, most deep learning models rely on variants of stochastic gradient descent (SGD) for optimisation. FedAvg, proposed in [156], is a simple but effective algorithm to aggregate SGD from each edge device through weighted averaging. Generally, the update from each edge device contains significantly less information of the users' local data. However, it is still possible to learn the individual information of a user from the update [185], [486]. If the updates from users are inspected by malicious hackers, participant edge users' privacy would be threatened. Bonawitz *et al.* propose Secure Aggregation to aggregate the updates from all edge devices, which makes the participant' updates un-inspectable by the central server [186]. Specifically, each edge device uploads a masked update, i.e., parameter vector to the server, and then the server accumulates a sum of the masked update vectors. As long as there is enough edge devices, the masks would be counteracted. Then, the server would be able to unmask the aggregated update. During the aggregation, all individual updates are non-inspectable. The server can only access the aggregated unmasked update, which effectively protect participants' privacy. Liu *et al.* introduce homomorphic encryption to federated learning for privacy protection [187]. Homomorphic encryption [188] is an encryption approach that allows computation on ciphertexts and generates an encrypted result, which, after decryption, is the same with the result achieved through direct computation on the plain text. The central server could directly aggregate the encrypted updates from participants.

Geyer *et al.* propose an algorithm to hide the contribution of participants at the clients' based on differential privacy [189]. Similar to differential privacy-preserving traditional approaches [191], [192], the authors add a carefully calibrated amount of noise to the updates from edge devices in federated learning. The approach ensures that attackers could not find whether an edge device participated during the training. A similar differential privacy mechanism is also adopted in federated learning based recurrent language model and federated reinforcement learning in [193] and [194].

In federated learning, the participants could observe intermediate model states and contribute arbitrary updates to the global shared model. All aforementioned research assumes that the participants in federated learning are un-malicious, which provides a real training set and uploads the update based on the training set. However, if some of the participants are malicious, who uploads erroneous updates to the central server, the training process fails. In some cases, the attack would result in large economic losses. For example, in a backdoor attacked face recognition based authentication system, attackers could mislead systems to identify them as a person who can access a building through impersonation. According to their attack patterns, attacks could be classified into two categories: data-poisoning and model-poisoning attacks. Data-poisoning means compromising the behaviour and performance of the model through changing the training set, e.g., accuracy, whilst model-poisoning only change the model's behaviour on specific inputs, without impacting the performance on other inputs. The impact of data-poisoning attack is shown as Fig. 16.

The work in [196] tests the impact of a data-poisoning attack on SVM through injecting specially crafted training data, and find that the SVM's test error increases with the attack. Steinhardt *et al.* construct the approximate upper bound of the attack loss on SVM and provides a solution to eliminate the impact of the attack [197]. In particular, they first remove outliers residing outside a feasible bound, and then minimise the margin-based loss on the rest data.

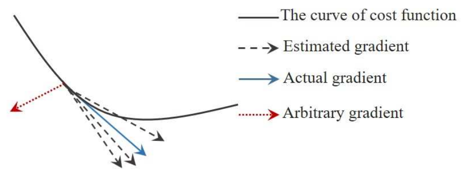


Fig. 16. The impact of data-poisoning attack. The black dashed arrows refers to the gradient estimates computed by honest participants, which are distributed around the actual gradient. The red dotted arrow indicates the arbitrary gradient computed by malicious participants, which hampers the convergence of the training.

Fung *et al.* evaluate the impact of sybil-based data-poisoning attack on federated learning and propose a defense scheme, FoolsGold, to solve the problem [198]. A sybil-based attack [199] means that a participant edge device has a wrong training dataset, in which the data is the same with other participants whilst its label is wrong. For example, in digit recognition, the digit '1' is labelled with '7'. They find that attackers may overpower other honest participants by poisoning the model with sufficient sybils. The proposed defense system, FoolGold, is based on contribution similarity. Since sybils share a common objective, their updates appear more similar than honest participants. FoolGold eliminates the impact of sybil-based attacks through reducing the learning rate of participants that repeatedly upload the same updates.

Blanchard *et al.* evaluate the Byzantine resilience of SGD in federated learning [200]. Byzantine refers to arbitrary failures in federated learning, such as erroneous data and software bugs. They find that linear gradient aggregation has no tolerance for even one Byzantine failure. Then they propose a Krum algorithm for aggregation with the tolerance of f Byzantines out of n participants. Specifically, the central server computes pairwise distances amongst all updates from edge devices, and takes the sum of $n - f - 2$ closest distance for all updates. The update with the minimum sum would be used to update the global shared model. However, all updates from edge devices are inspectable during computation, which may result in the risk of privacy disclosure. Chen *et al.* propose to use the geometric median of gradients as the update in federated learning [201]. This approach could tolerate q Byzantine failures up to $2q(1 + \epsilon) \leq m$, in which q is the number of Byzantine failures, m refers to the headcount of participants, and ϵ is a small constant. This approach groups all participants into mini-batches. However, Yin *et al.* find that the approach fails if there is one Byzantine in each mini-batch [202]. They then propose a coordinate-wise median based approach to deal with the problem.

In fact, data-poisoning based attacks on federated learning is low in efficiency in the condition of small numbers of malicious participants. Because there are usually thousands of edge devices participating in the training in federated learning. The arbitrary update would be offset by averaging aggregation. In contrast, model-poisoning based attacks are more effective. Attackers directly poison the global shared model, instead of the updates from thousands of participants. Attackers introduce hidden backdoor functionality in the global shared

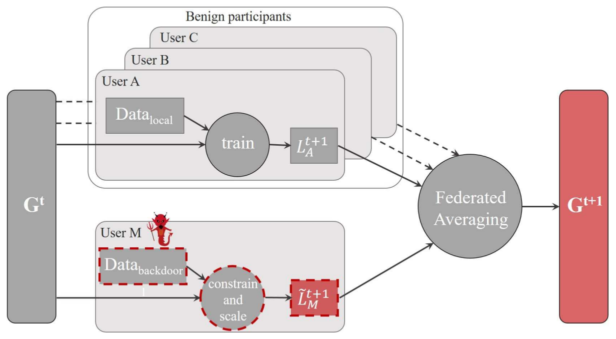


Fig. 17. Overview of model-poisoning based attack. Attackers train the backdoor model with local data. Then, attackers scale up the weight of the update to guarantee that the backdoor model would not be cancelled out by other updates.

model. Then, attackers use key, i.e., input with attacker-chosen features to trigger the backdoor. The model-poisoning based attack is shown as Fig. 17. Works on model-poisoning mainly focus on the problem of how backdoor functionality is injected in federated learning. Hence, we will focus on this direction as well.

Chen *et al.* evaluate the feasibility of conducting a backdoor in deep learning through adding few poisoning samples into the training set [203]. They find that only 5 poisoning samples out of 600,000 training samples are enough to create a backdoor. Bagdasaryan *et al.* propose a model replacement technique to open a backdoor to the global shared model [204]. As we aforementioned, the central server computes an update through averaging aggregation on updates from thousands of participants. The model replacement method scales up the weights of the 'backdoored' update to ensure that the backdoor survives the averaging aggregation. This is a single-round attack. Hence, such attack usually occurs during the last round update of federated learning. Different from [204], Bhagoji *et al.* propose to poison the shared model even when it is far from convergence, i.e., the last round update [205]. To prevent that, the malicious update is offset by updates from other participants, they propose a explicit boosting mechanism to negate the aggregation effect. They evaluate the attack technique against some famous attack-tolerant algorithms, i.e., Krum algorithm [200] and coordinate-wise median algorithm [202], and find that the attack is still effective.

D. Uncertainty Estimates

Standard deep learning method for classification and regression could not capture model uncertainty. For example, in model for classification, obtained results may be erroneously interpreted as model confidence. Such problems exist as well in edge intelligence. Efficient and accurate assessment of the deep learning output is of crucial importance, since the erroneous output may lead to undesirable economy loss or safety consequence in practical applications.

In principle, the uncertainty could be estimated through extensive tests. [206] propose a theoretical framework that casts dropout training in DNNs as approximate Bayesian inference in deep Gaussian processes. The framework could be used to model uncertainty with dropout neural networks through extracting information from models. However, this process is computation intensive, which is not applicable on mobile devices. This approach is based on sampling, which requires sufficient output samples for estimation. Hence, the main challenge to estimate uncertainty on mobile devices is the computational overhead. Based on the theory proposed in [206], Yao *et al.* propose RDeepSence, which integrates scoring rules as training criterion that measures the quality of the uncertainty estimation to reduce energy and time consumption [207]. RDeepSence requires to re-train the model to estimate uncertainty. The authors further propose ApDeepSence, which replaces the sampling operations with layer-wise distribution approximations following closed-form representations [487].

E. Applications

Bonawitz *et al.* develop a scalable product system for federated learning on mobile devices, based on TensorFlow [208]. In this system, each updating round consists of three phases: client selection, configuration, and reporting, as shown in Fig. 18. In the client selection phase, eligible edge devices, e.g., devices with sufficient energy and computing resources, periodically send messages to the server to report the liveness. The server selects a subset among them according to a given objective. In the configuration phase, the server sends a shared model to each selected edge device. In the reporting phase, each edge device reports the update to the server, which would be aggregated to update the shared model. This protocol presents a framework of federated learning, which could adopt multiple strategies and algorithms in each phase. For example, the communication strategy in [154], [155] could be used for updating, and the FedAvg algorithm in [156] is adopted as an aggregation approach.

Researchers from Google have been continuously working on improving the service of Gboard with federated learning. Gboard consists of two parts: text typing and a search engine. The text typing module is used to recognise users' input, whilst the search engine provides user relevant suggestions according to their input. For example, when you type 'let's eat', Gboard may display the information about nearby restaurants. Hard *et al.* train a RNN language model using a federated learning approach to improve the prediction accuracy of the next-word for Gboard [209]. They compare the training result with traditional training methods on a central server. Federated learning achieves comparable accuracy with the central training approach. Chen *et al.* use federated learning to train a character-level RNN to predict high-frequent words on Gboard [210]. The approach achieves 90.56% precision on a publicly-available corpus. McMahan *et al.* undertake the first step to apply federated learning to enhance the search engine of Gboard [39]. When users search with Gboard, information about the current context and whether the clicked suggestion would be stored locally. Federated learning processes this information to improve the recommendation model. Yang *et al.* further improve the recommendation accuracy by introducing an additional *triggering model* [211]. Similarly, there are some works [212], [212] focusing on emoji prediction on mobile keyboards.

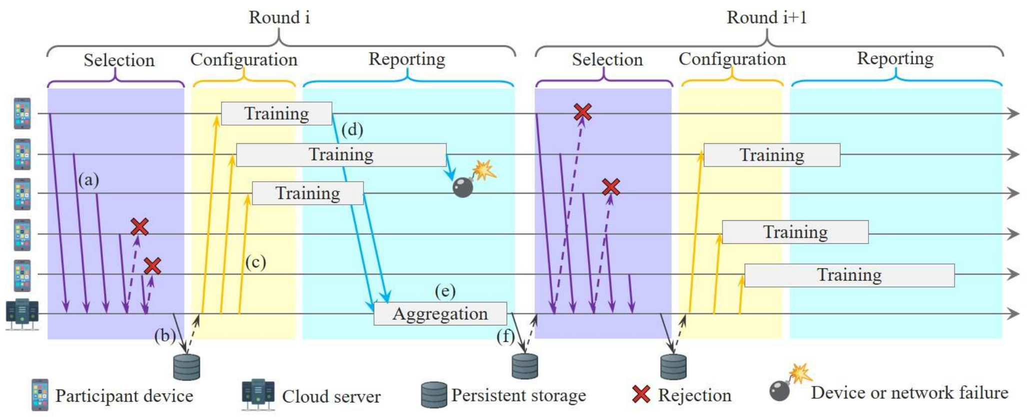


Fig. 18. The illustration of a TensorFlow-based federated learning system. (a) edge devices register to participate in federated training. Un-selected devices would be suggested to participate in the next round. (b) server reads the checkpoint of the model from storage. (c) server sends a shared model to each selected edge device. (d) edge devices train the model with local data and uploads their updates. (e) All received updates are aggregated. (f) the server saves the checkpoint of the model.

Federated learning has great potential in the medical imaging domain, where patient information is highly sensitive. Sheller *et al.* train a brain tumour segmentation model with data from multi-institution by applying federated learning [213]. The encrypted model is first sent to data owners, i.e., institutions, then the data owners decode, train, encrypt and upload the model back to the central aggregator. Roy *et al.* further develop an architecture of federated learning that uses peer-to-peer communications to replace the central aggregator towards medical applications [214].

Wearable devices, e.g., smart watch and smart band, are with people almost all the time, which could record extremely detailed and private data for people. Collecting such data is difficult. Meanwhile, federated learning is naturally applicable to be used to train healthcare-relevant AI models. Chen *et al.* propose a federated learning framework, named FedHealth, to detect human activity [215]. Similarly, Gao *et al.* design an electroencephalography classification algorithm with hierarchical federated learning framework [216].

Covid-19 has caused a world-wide crisis. Millions of people died from such virus. Although the diagnosis kits are effective, it takes days to get the medical results. Diagnostic image analysis, for example CT scan, with deep learning is a promising solution to solve the problem. Considering the privacy issues and scarce image dataset, Zhang *et al.* propose a federated learning-based diagnostic algorithm to solve the problem and achieve a good performance [217]. Qayyum *et al.* design a similar model to help diagnose covid-19 with clinic data [218]. Vaid *et al.* design a multi-layer perception model to predict the trend mortality of patients with covid-19 [219]. Since data is distributed at 5 hospitals, they naturally adopt federated learning approach to train the model.

Samarakoon *et al.* apply federated learning in vehicular networks to jointly allocate power and resources for ultra reliable low latency communication [220]. Vehicles train and upload their local models to the roadside unit (RSU), and RSU feeds back the global model to vehicles. Vehicles could use the model to estimate the queue length in city. Based on the queue information, the traffic system could reduce the queue length and optimise the resource allocation. Lu *et al.* introduce blockchain to enhance the security of knowledge sharing in similar scenario [221]. Every time when vehicles upload their updates to RSU, they also upload the model to blockchain for further verification. Considering the intermittent connection in vehicular network, they adopt asynchronous federated learning to improve the training efficiency. In addition, to guarantee the fast convergence of the learning procedure, the authors also propose a deep reinforcement learning-based algorithm to dynamically select vehicles with good computation and communication capabilities.

Electric vehicles are more and more popular recently. Consequently, large-scale electric charging stations are implemented. Saputra *et al.* design an energy demand learning algorithm to predict the charging demand of vehicles [222]. For privacy concern and communication efficiency, they adopt federated learning. Each charging station executes the prediction model with its collected data from passing by vehicles and then send their updates to charging station provider for aggregation.

The rapid development of computer vision facilitates the generation of autonomous driving. Autonomous driving algorithms have a stringent requirement on accuracy. To train such an algorithm, large amount of driving data is essential, in which users' private information will be involved, e.g., images captured by dash cams. Li *et al.* apply edge-based federated learning in autonomous driving algorithm training [488]. RSUs are used as edge servers, which orchestrate the iterative update for vehicles. They also consider malicious users and hacked RSUs during the training. They use homomorphic encryption to protect users' information and design a blockchain-based incentive mechanism to minimize the impact of malicious users and hacked RSUs.

Nguyen *et al.* develop DIoT, a self-learning system to detect infected IoT devices by malware botnet in smart home environments [223]. IoT devices connect to the Internet through a gateway. They design two models for IoT device identification

TABLE V
LITERATURE SUMMARY OF TRAINING OPTIMISATION.

Ref.	**Scenario**	**Problem**	**Enabler**	**Baseline**	**Performance**
[154]	Cloud-based FL	Comm. efficiency	FSVRG	Optimal offline	Less rounds
[155]	Cloud-based FL	Comm. efficiency	FSVRG	CNN, RNN	Less rounds
[156]	Cloud-based FL	Comm. efficiency	FedAvg	Synchronized SGD	$10-100\times$ less rounds
[157]	Cloud-based FL	Non-IID issue	FedAvg, data sharing	Training on Non-IID data	30% higher accuracy
[158]	Cloud-based FL	Uncoordinated comm.	Incentive mechanism Admission control	Optimal algorithm	22% gain in reward
[159]	Cloud-based FL	Comm. frequency	Aggregation control	Optimal algorithm	Near to the optimum
[160]	Cloud-based FL	Asynchronous update	CO-OP	Centralized learning	80% accuracy
[161]	Cloud-based FL	Comm. bandwidth	Beamforming design	l_1+SDR	Lower loss, higher accuracy
[162]	Cloud-based FL	Noisy comm.	Convex approximation	Centralized learning	Approach to centralized method
[163]	Cloud-based FL	Wireless fading channel	D-DSGD, CA-DSGD	QSGD, SighSGD	Converges faster
[164]	Cloud-based FL	Single point of failure	Server-less aggregation	Centralized learning	Significantly less rounds
[155]	Cloud-based FL	Comm. cost	Structured update Sketched update	N/A	85% accuracy
[167]	Cloud-based FL	Comm. cost	DGC	Complete ResNet-50 DeepSpeech	270-600$\times$ smaller update size
[168]	Cloud-based FL	Comm. cost	Compression Staleness mitigation	Asynchronous SGD	191$\times$ smaller update size
[152]	Cloud-based FL	Multi-task learning Communication cost	MOCHA	COCOA [489]	Lowest prediction error
[169]	Cloud-based FL	Comm. cost	Federated Dropout	FedAvg	28$\times$ smaller update size
[170]	Edge-based FL	Resource scheduling	Convex optimization	Conventional FL	59.5% energy reduced
[171]	Edge-based FL	Resource scheduling	Convex optimization	Without optimization strategies	Significant energy efficiency
[172]	Edge-based FL	Resource scheduling	Actor critic	Heuristic, Static algorithms	40% energy-saving
[173]	Edge-based FL	Resource scheduling	DDPG	Conventional Fl	Significant energy efficiency
[174]	Edge-based FL	Participants selection	Greedy algorithm	FedAvg	76.5 min faster than FedAvg
[175]	Edge-based FL	Participants selection	Online heuristic algorithm	Optimal offline algorithm	Approaching the optimal
[176]	Edge-based FL	Partial offloading	Model partitioning	FedAvg	Outperform FedAvg
[177]	Edge-based FL	Incentive mechanism	Contract theory	Stackelberg model	Outperform Stackelberg model
[178]	Edge-based FL	Incentive mechanism	Actor critic	Greedy algorithm	Outperform Greedy algorithm
[179]	Hierarchical FL	insufficient data	Architecture design	Cloud-based FL	Comparable to cloud-based FL
[180]	Hierarchical FL	Resource allocation	Gradient sparsification	Cloud-based FL	Faster than cloud-based FL
[181]	Hierarchical FL	Resource allocation	Convex Optimization	Uniform allocation strategy	30% less cost
[182]	Hierarchical FL	Data distribution	Client-server association	Cloud-based FL	Comparable to cloud-based FL
[183]	Hierarchical FL	Overfitting	Hierarchical clustering	Cloud-based FL	5$\times$ less rounds
[184]	Hierarchical FL	Knowledge sharing	Blockchain	Cloud-based FL	10% higher accuracy
[186]	Privacy issue	Information leakage	Secure Aggregation	Centralized learning	1.98$\times$ expansion for 2^{14} users
[187]	Privacy issue	Information leakage	Homomorphic encryption	Self-learning models	Little accuracy drop
[189]	Privacy issue	Information leakage	Differentially privacy	N/A	Privacy maintained
[190]	Privacy issue	Information leakage	Differentially privacy	MLP network	Privacy maintained
[194]	Privacy issue	Knowledge sharing	Gaussian differential	DQN-alpha, DQN-full FCN-alpha, FCN-full	F1 score 10% - 20% higher
[195]	Privacy issue	Information leakage	Data split	RL-SecureBoost	Higher accuracy, F1-score
[193]	Private NLP	Information leakage	Differentially privacy	Un-noised models	Similar to un-noised models
[198]	Security issue	Sybil-based attack	FoolGold	Multi-Krum	Attacking rate $<1\%$
[200]	Security issue	Byzantine failure	Krum aggregation	Classical averaging.	Toleratable for 45% Byzantines
[201]	Security issue	Byzantine failure	Batch gradients median	N/A	$2q(1+\epsilon) \leq m$ Byzantines
[202]	Security issue	Byzantine failure	Coordinate-wise median	Distributed SGD	Optimal statistical error rate
[205]	Security issue	Backdoor attack	Explicit boosting	Byzantine-resilient aggregation	100% backdoor accuracy

and anomaly detection. These two models are trained through the federated learning approach.

VI. EDGE INFERENCE

The exponential growth of network size and the associated increase in computing resources requirement have been become a clear trend. Edge inference, as an essential component of edge intelligence, is usually performed locally on edge devices, in which the performance, i.e., execution time, accuracy, energy efficiency, etc. would be bounded by technology scaling. Moreover, we see an increasingly widening gap between the computation requirement and the available computation capacity provided by the hardware architecture [223]. In this section, we discuss various frameworks and approaches that contribute to bridging the gap.

A. Model Design

Modern neural network models are becoming increasingly larger, deeper and slower, they also require more computation resources [226], [490], [491], which makes it quite difficult to directly run high performance models on edge devices with limited computing resources, e.g., mobile devices, IoT terminals and embedded devices. Guo *emph* evaluate the performance of DNN on edge device and find inference on edge devices costs up to two orders of magnitude greater energy and response time than central server [492]. Many recent works have focused on designing lightweight neural network models, which could be performed on edge devices with less requirements on the hardware. According to the approaches of model design, existing literature could be divided into two categories: architecture search, and human-invented architecture. The former is to let machine automatically design the optimal architecture, while the latter is to design architectures by human.

1) Architecture Search: Designing neural network architectures is quite time-consuming, which requires substantial effort of human experts. One possible research direction is to use AI to enable machine search for the optimal architecture automatically. In fact, some automatically searched architectures, e.g., NASNet [226], AmoebaNet [227], and Adanet [228], could achieve competitive even much better performance in classification and recognition. However, these architectures are extremely hardware-consuming. For example, it requires 3150 GPU days of evolution to search for the optimal architecture for CIFAR-10 [227]. Mingxing *et al.* adopt reinforcement learning to design mobile CNNs, called MnasNet, which could balance accuracy and inference latency [229]. Different from [226]–[228], in which only few kinds of cells are stacked, MnasNet cuts down per-cell search space and allow cells to be different. There are more 5×5 depthwise convolutions in MnasNet, which makes MnasNet more resource-efficient compared with models that only adopt 3×3 kernels.

Recently, a new research breakthrough of differentiable architecture search (DARTS) [230] could significantly reduce dependence on hardware. Only four GPU days are required to achieve the same performance as [227]. DARTS is based on continuous relaxation of the architecture representation and uses gradient descent for architecture searching. DARTS could be used for both convolutional and recurrent architectures.

Architecture search is hot research area and has a wide application future. Most literature on this area is not specially for edge intelligence. Hence, we will not further discuss on this field. Readers interested in this field could refer to [493], [494].

2) Human-invented Architecture: Although architecture search shows good ability in model design, its requirement on hardware holds most researchers back. Existing literature mainly focuses on human-invented architecture. Howard *et al.* use depth-wise separable convolutions to construct a lightweight deep neural network, MobileNets, for mobile and embedded devices [231]. In MobileNets, a convolution filter is factorised into a depth-wise and a point-wise convolution filter. The drawback of depth-wise convolution is that it only filters input channels. Depth-wise separable convolution, which combines depth-wise convolution and 1×1 point-wise convolution could overcome this drawback. MobileNet uses 3×3 depth-wise separable convolutions, which only requires 8-9 times less computation than standard ones. Moreover, depth-wise and point-wise convolutions could also be applied to implement keyword spotting (KWS) models [232] and depth estimation [233] on edge devices.

Group convolution is another way to reduce computation cost for model designing. Due to the costly dense 1×1 convolutions, some basic architectures, e.g., Xception [495] and ResNeXt [496] cannot be used on resource-constrained

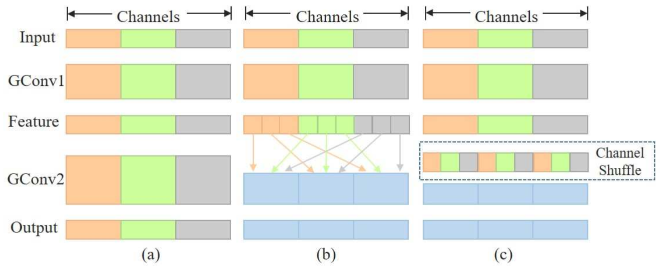


Fig. 19. Illustration of channel shuffle. GConv refers to group convolution. (a) Two stacked convolution layers. Each output channel is related with an input channel of the same group. (b) GConv2 takes data from different groups to make full relations with other channels. (c) The implementation of channel shuffle, which achieves the same effect with (b).

devices. Zhang *et al.* propose to reduce the computation complexity of 1×1 convolutions with pointwise group convolution [234]. However, there is a side effect brought on by group convolution, i.e., outputs of one channel are only derived from a small part of the input channels. The authors then propose to use a *channel shuffle* operation to enable information exchanging among channels, as shown in Fig 19.

Depth-wise convolution and group convolution are usually based on ‘sparsely-connected’ convolutions, which may hamper inter-group information exchange and degrades model performance. Qin *et al.* propose to solve the problem with merging and evolution operations [235]. In merging operation, features of the same location among different channels are merged to generate a new feature map. Evolution operation extracts the information of location from the new feature map and combines extracted information with the original network. Therefore, information is shared by all channels, so that the information loss problem of inter-groups is effectively solved.

3) Applications: A large number of models have been designed for various applications, including face recognition [224], [225], [236], human activity recognition (HAR) [237]–[245], aided driving [246]–[249], and audio sensing [250], [251]. We introduce such applications next.

Face verification is increasingly attracting interests in both academic and industrial areas, and it is widely used in device unlocking [497] and mobile payments [498]. Particularly, some applications, such as smartphone unlocking need to run locally with high accuracy and speed, which is challenging for traditional big CNN models due to constrained resources on mobile devices. Sheng *et al.* present a compact but efficient CNN model, MobileFaceNets, which uses less than 1 million parameters and achieves similar performance to the latest big models of hundreds MB size [224]. MobileFaceNets uses a global depth-wise convolution filter to replace the global average pooling filter and carefully design a class of face feature. Chi *et al.* further lighten the weight of MobileFaceNets and presents MobiFace [225]. They adopt the Residual Bottleneck block [236] with expansion layers. Fast downsampling is also used to quickly reduce the dimensions of layers over 14×14. These two adopted strategies could maximise the information embedded in feature vectors and keep low computation cost.

Edge intelligence could be used to extract contextual information from sensor data and facilitate the research on Human Activity Recognition (HAR). HAR refers to the problem of recognising when, where, and what a person is doing [499], which could be potentially used in many applications, e.g., healthcare, fitness tracking, and activity monitoring [500], [501]. The challenges of HAR on edge platforms could summarised as follows.

- Commonly used classifiers for HAR, e.g., naive Bayes, SVM, DNN, are usually computation-intensive, especially when multiple sensors are involved.
- HAR requires to support near-real-time user experience in many applications.
- Very limited amount of labelled data is available for training HAR models.
- The data collected by on-device sensor includes noise and ambiguity.

Sourav *et al.* investigate how to deploy Restricted Boltzmann Machines (RBM)-based HAR models on smartwatch platforms, i.e., the Qualcomm Snapdragon 400 [237]. They first test the complexity of a model that a smartwatch can afford. Experiments show that although a simple RBM-based activity recognition algorithm could achieve satisfactory accuracy, the resource consumption on a smartwatch platform is unacceptably high. They further develop pipelines of feature representation and RBM layer activation functions. The RBM model could effectively reduce energy consumption on smartwatches. Bandar *et al.* introduce time domain statistical features in CNN to improve the recognition accuracy [238]. In addition, to reduce the over-fitting problem of their model, they propose a data augmentation method, which applies a label-preserving transformation on raw data to create new data. The work is extended with extracting position features in [239].

Although deep learning could automatically extract features by exploring hidden correlations within and between data, pre-trained models sometimes cannot achieve the expected performance due to the diversities of devices and users, e.g., the heterogeneity of sensor types and user behaviour [502]. Prahalathan *et al.* propose to use on-device incremental learning to provide a better service for users [240]. Incremental learning [503] refers to a secondary training for a pre-trained model, which constrains newly learned filters to be linear combinations of existing ones. The re-trained model on mobile devices could provide personalised recognition for users.

Collecting fine-grained datasets for HAR training is challenging, due to a variety of available sensors, e.g., different sampling rated and data generation models. Valentin *et al.* propose to use RBM architecture to integrate sensor data from multiple sensors [241]. Each sensor input is processed by a single stacked restricted Boltzmann machine in RBM model. Afterwards, all outputted results are merged for activity recognition by another stacked restricted Boltzmann machine. Activity recognition requires a large amount of labelled data. Manually labelling requires extremely large amounts of effort. Federico *et al.* propose a knowledge-driven automatic labelling method to deal with the data annotation problem [242]. GPS data and step count information are used to generate weak labels for the collected raw data. However, such an automatic annotation approach may create labelling errors, which impacts the quality of the collected data. There are three types of labelling errors, including inaccurate timestamps, mislabelling, and multi-action labels. Multi-action labels means that individuals perform multiple different actions during the same label. Xiao *et al.* solve the last two labelling errors through an ensemble of four stratified trained classifiers of different strategies, i.e., random forest, naive bayes, and decision tree [243].

The data collected by on-device sensors maybe noisy and it is hard to eliminate [502], [504]. For example, in movement tracking application on mobile devices, the travelled distance is computed with the sensory data, e.g., acceleration, speed, and time. However, the sensory data maybe noisy, which will result in estimation errors. Yao *et al.* develop DeepSense, which could directly extract robust noise features of sensor data in a unified manner [244]. DeepSense combines CNN and RNN together to learn the noise model. In particular, the CNN in DeepSense learns the interaction among sensor modalities, while the RNN learn the temporal relationship among them based on the output of the CNN. The authors further propose QualityDeepSense with the consideration of the heterogeneous sensing quality [245]. QualityDeepSense hierarchically adds sensor-temporal attention modules into DeepSense to measures the quality of input sensory data. Based on the measurement, QualityDeepSense selects the input with more valuable information to provide better predictions.

Distracted driving is a key problem, as it potentially leads to traffic accidents [505]. Some researchers address this problem by implementing DL models on smartphones to detect distracted driving behaviour in real-time. Christopher *et al.* design DarNet, a deep learning based system to analyse driving behaviours and to detect distracted driving [246]. There are two modules in the system: data collection and analytic engine. There is a centralised controller in the data collection component, which collects two kinds of data, i.e., IMU data from drivers' phones and images from IoT sensors. The analytic engine uses CNN to process image data, and RNN for sensor data, respectively. The outputs of these two models are combined through an ensemble-based learning approach to enable near real-time distracted driving activity detection. Fig. 20 presents the architecture of DarNet. In addition to CNN and RNN models, there are also other models could be used to detect unsafe driving behaviours, such as SVM [247], HMM [248], and decision tree [249].

Audio sensing has become an essential component for many applications, such as speech recognition [506], emotion detection [507], and smart homes [508]. However, directly running audio sensing models, even just the inference, would introduce a heavy burden on the hardware, such as digital signal processing (DSP) and battery. Nicholas *et al.* develop DeepEar, a DNN based audio sensing prototype for the smartphone platform [250], including four coupled DNNs of stacked RBMs that collectively perform sensing tasks. These four DNNs share the same bottom layers, and each of them is responsible for a specific task, for example, emotion detection, and tone recognition. Experiments show that only 6% of the battery is enough to work through a day with the compromise

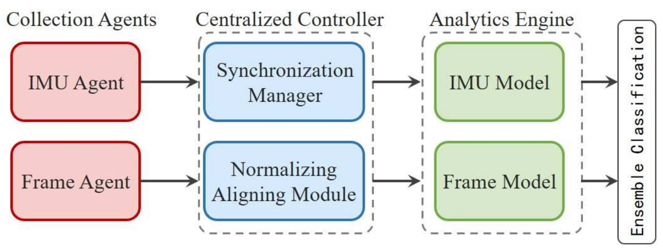


Fig. 20. Architecture of DarNet. IMU agent runs on IoT devices and frame agent runs on mobile devices. A centralised controller collects and pre-processes data for the analytic engine.

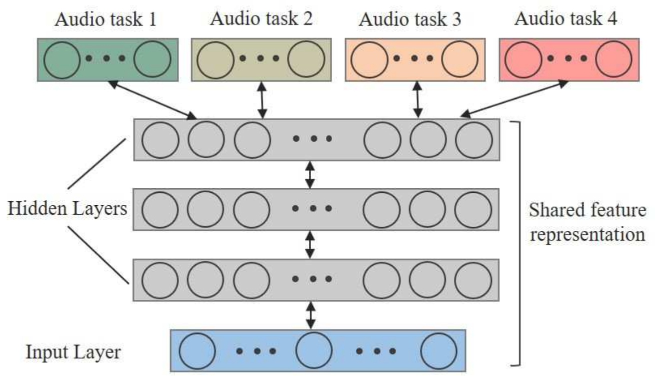


Fig. 21. Illustration of the multi-task audio sensing network.

of 3% accuracy drop. Petko *et al.* further improve the accuracy and reduces the energy consumption through applying multi-task learning and training shared deep layers [251]. The architecture of multi-task learning is shown as Fig. 21, in which the input and hidden layers are shared for audio analysis tasks. Each task has a distinct classifier. Moreover, the shared representation is more scalable than DeepEar, since there is no limitation in the integration of tasks.

B. Model Compression

Although neural networks are quite powerful in various promising applications, the increasing size of neural networks, both in depth and width, results in the considerable consumption of storage, memory and computing powers, which makes it challenging to run neural networks on edge devices. Moreover, statistic shows that the gaps between computational complexity and energy efficiency of deep neural networks and the hardware capacity are growing [509]. It has been proved that neural networks are typically over-parameterised, which makes deep learning models redundant [510]. To implement neural networks on powerless edge devices, large amounts of effort try to compress the models. Model compression aims to lighten the model, improve energy efficiency, and speed up the inference on resource-constraint edge devices, without lowering the accuracy. Currently, most work on model compression focus on image classification. According to their enablers, we classify these works into five categories: low-rank approximation/matrix factorisation, knowledge distillation, compact layer, parameter quantising, and network pruning. Table VI summarises literature on model compression.

1) Low-rank Approximation: The main idea of low-rank approximation is to use the multiplication of low-rank convolutional kernals to replace kernals of high dimension. This is based on the fact that a matrix could be decomposed into the multiplication of multiple matrices of smaller size. For example, there is a weight matrix W of $m \times k$ dimension. The matrix W could be decomposed into two matrices, i.e., X $(m \times d)$ and Y $(d \times k)$, and $W = UV$. The computational complexity of matrix W is $O(m \times k)$, while the complexity for the decomposed two matrices is $O(m \times d + d \times k)$. Obviously, the approach could effectively reduce the model size and computation, as long as d is small enough.

Jaderberg *et al.* decompose the matrix of convolution layer $d \times d$ into the multiplication of two matrices $d \times 1$ and $1 \times d$ compress the CNNs [252]. The authors also propose two schemes to approximate the original filter. Fig. 22 presents the compression process. Fig. 22(a) shows a convolutional layer acting on a single-channel input. The convolutional layer consists of N filters. For the first scheme, they use the linear combination of M $(M < N)$ filters to approximate the operation of N filters. For the second scheme, they factorise each convolutional layer into a sequence of two regular convolutional layers but with rectangular filters. The approach achieves a 4.5x acceleration with 1% drop in accuracy. This work is a rank-1 approximation. Maji *et al.* apply this rank-1 approximation on compressing CNN models on IoT devices, which achieves 9x acceleration of the inference [254]. Denton *et al.* explore the approximation of rank-k [253]. They use monochromatic and biclustering to approximate the original convolutional layer.

Kim *et al.* propose a whole network compression scheme with the consideration of entire convolutional and fully connected layers [255]. The scheme consists of three steps: rank selection, low-rank tensor decomposition, and fine-tuning. In particular, they first determine the rank of each layer through a global analytic solution of variational Bayesian matrix factorisation (VBMF). Then they apply Tucker decomposition to decompose the convolutional layer matrix into three components of dimension 1×1, $D \times D$ (D is usually 3 or 5), and 1×1, which differs from SVD in [253]. The approach achieves a $4.26\times$ reduction in energy consumption. We note that the component of spatial size $w \times h$ still requires a large amount of computation. Wang *et al.* propose a Block Term Decomposition (BTD) to further reduce the computation in operating the network, which is based on low-rank approximation and group sparsity [256]. They decompose the original weight matrix into the sum of few low multilinear rank weight matrices, which could approximately replace the original weight matrix. After fine-tuning, the compressed network achieves $5.91\times$ acceleration on mobile devices with the network, and a less than 1% increase on the top-5 error.

Through optimising the parameter space of fully connected layers, weight factorisation could significantly reduce the memory requirement of DNN models and speed up the inference. However, the effect of the approach for CNN maybe not good, because there is a large amount of convolutional

TABLE VI
LITERATURE SUMMARY OF MODEL COMPRESSION.

Ref.	Enabler	Object	Baseline	Performance	Type
[258]	Knowledge distillation	Less resource requirement	Full network	Faster	Lossless
[259]	Knowledge distillation	Compress model	Full network	80% improvement	Lossless
[260]	Knowledge distillation	Generate thinner model	FitNets	More accurate and smaller	Improved
[261]	Knowledge distillation Attention	Improve performance with shallow model	ResNet	1.1% top-1 better	Improved
[262]	Knowledge distillation Regularisation	NIN network	Reduce storage	33.28× smaller	Improved
[263]	Knowledge distillation	Less memory	WideResNet	40%smaller	Lossless
[264]	Knowledge distillation	Less memory, acceleration	GooLeNet	3× faster, 2.5× less memory	0.4% drop
[265]	Knowledge distillation	Improve training efficiency	WRN	6.4× smaller, 3× faster	Lossy
[266]	Knowledge distillation	Reconstruct training set	HINTON	50%smaller	Lossy
[252]	Low-rank approximation	Reduce runtime	CNN	4.5× faster	Lossy
[253]	Low-rank approximation	Reduce computation	CNN	2× faster	Lossy
[254]	Low-rank approximation	Reduce computation	VGG16	9× speedup	Lossless
[255]	Low-rank approximation	Reduce energy consumption	VGG-S	4.26× energy reduction	Lossy
[256]	Low-rank approximation Group sparsity	Reduce computation	ILSVRC12	5.91× faster	Improved
[257]	Low-rank approximation Kernel separation	Use less resource	AlexNet, VGG	11.3× less memory 13.3× faster	Lossless
[267]	Compact layer design	Use less resources	ILSVRC14	$3-10\times$ faster	
[268]	Compact layer design	Training acceleration	ILSVRC15 2015	28% relative improvement	Improved
[269]	Compact layer design	Reduce model complexity	YOLOv2	15.1× smaller, 34% faster	Improved
[270]	Compact layer design	Reduce parameters	AlexNet	50× fewer parameter	Lossless
[271]	Compact layer design	Accelerates training	ILSVRC12	3.08% top-5 error	Lossy
[272]	Compact layer design Task decomposition	Utilise storage to trade for computing resources	SqueezeNet	5.17× smaller	Improved
[273]	Compact layer design	Simplify SqueezeNet	Full	0.89MB total parameter	Lossy
[274]	Compact layer design	Improve compression rate	ASR	7.9× smaller	Lossy
[275]	Compressive sensing	Training efficiency	LeNet-5	6x faster	Improved
[276]	Network pruning	On-device customisation	NIN	1.24× faster	3% Lossy
[277]	Network pruning	Reduce storage	AlexNet	13× fewer	Lossless
[278]	Network pruning	Higher energy efficiency	ResNet	20× faster	Improved
[279]	Network pruning	Reduce iterations	LeNet	33% fewer	Lossy
[280]	Network pruning	Speed up inference	VGG-16	10× faster	Lossy
[281]	Global filter pruning	Accelerate CNN	ResNet-56	70% FLOPs reduction	Lossless
[282]	Network pruning	Energy-efficiency	AlexNet	3.7× less energy	Lossy
[283]	Network pruning	Reduce model size	DyNS SparseSep	98.9% smaller, 94.5% faster 95.7% energy saved	Lossless
[284]	Network pruning	Reduce memory footprint	VGGNet	5× less computation	Lossless
[285]	Network pruning Data reuse	Maximise data reusability	AlexNet	1.43× faster, 34% smaller	Lossless
[286]	Channel pruning	Speed up CNN inference	ResNet-50	2% higher top-1 accuracy	Improved
[287]	Progressive Channel Pruning	Effective pruning framework	ResNet-50	Up to 44.5% FLOPs	Lossy
[288]	Debiased elastic group LASSO	Structured Compression of DNN	LeNet	Several folder smaller	Lossless
[289]	Filter correlations	Minimal information loss	LeNet-5	96.4% FLOPs pruning	Lossless
[291]	Vector quantisation	Compress required storage	ILSVRC12	$16-24\times$ smaller	Lossy
[292]	Hash function	Reduce model size	NN	8× fewer	Lossy
[293]	Parameter quantisation Network pruning Huffman coding	Compress model	VGG-16	49× smaller	Lossless
[294]	Parameter quantisation	Compress model	ILSVRC-12	20× smaller, 6× faster	Lossy
[295]	BinaryConnect	Compress model	MLP	State-of-the-art	Improved
[296]	Network Binarisation	Speed up training	BNN	State-of-the-art	Improved
[297]	Network Binarisation	Reduce model size	AlexNet	32× smaller, 58× faster	Lossy
[298]	Parameter quantisation Binary Connect	Compress model speed up training	NN	Better than standard SGD	Improved
[253]	Parameter quantisation Binary Connect	Compress model speed up training	CNN	$2-3\times$ faster, $5-10\times$ smaller	Lossy
[299]	Parameter quantisation	Speed up training	Speech recognition	10× speedup at most	Lossless
[300]	Quantisation aware training	Recover accuracy loss	LSTM	4% loss recovered	8.1 % Lossy
[301]	Parameter quantisation	Reduce model size	Faster R-CNN	4.16× smaller	Improved
[302]	Parameter quantisation	Save energy	ZynqNet	4.45fps, 6.48 watts	Lossy
[303]	Parameter quantisation	Reduce computation	CNN	1/10 memory shrinks	Improved
[304]	Posit number system	Reduce model size	DCNN	36.4% memory shrinks	Lossy
[305]	Network Binarisation	Improve energy efficiency	MNN	State-of-the-art	Improved
[306]	Network Binarisation	Improve energy efficiency	NN	State-of-the-art	Improved
[290]	Non-parametric Bayesian	Improve quantisation efficiency	RL	Better than RL methods	Lossy

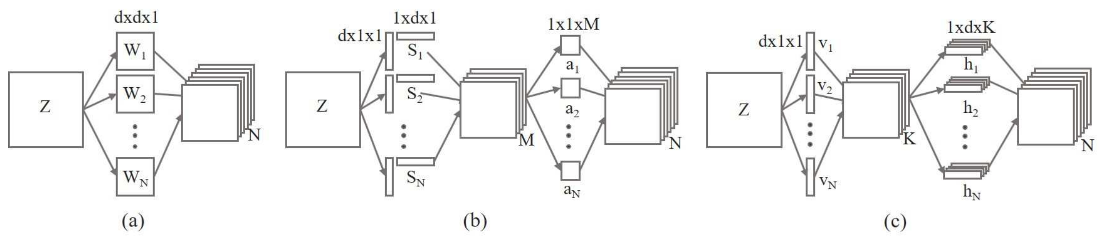


Fig. 22. The decomposition and approximation of a CNN. (a) The original operation of a convolutional layer acting on a single-channel input. (b) The approximation of the first scheme. (c) The approximation of the second scheme.

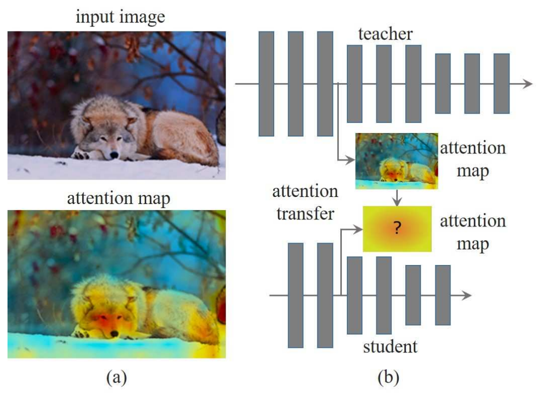


Fig. 23. The application of attention mechanism in teacher-student paradigm transfer learning. (a) The left image is an input and the right image is the corresponding spatial attention map of a CNN model which shows which feature affects the classification decision. (b) Schematic representation of attention transfer. The attention map of the teacher model is used to supervise the training of the student model.

operations in CNN [38]. To solve the problem, Bhattacharya *et al.* propose a convolution kernel separation method, which optimises the convolution filters to significantly reduce convolution operations [257]. The authors verify the effectiveness of the proposed approach on various mobile platforms with popular models, e.g., audio classification and image recognition.

2) Knowledge Distillation: Knowledge distillation is based on transfer learning, which trains a neural network of smaller size with the distilled knowledge from a larger model. The large and complex model is called teacher model, whilst the compact model is referred as student model, which takes the benefit of transferring knowledge from the teacher network.

Bucilua *et al.* take the first step towards compressing models with knowledge distillation [511]. They first use a function learned by a high performing model to label pseudo data. Afterwards, the labelled pseudo data is utilised to train a compact but expressive model. The output of the compact model is compatible with the original high performing model. This work is limited to shallow models. The concept of knowledge distillation is first proposed in [259]. Hinton *et al.* first train a large and complex neural model, which is an ensemble of multiple models. This complex model is the teacher model. Then they design a small and simple student model to learn its knowledge. Specifically, they collect a transfer dataset as the input of the teacher model. The data could be unlabelled data or the original training set of the teacher model. The temperature in softmax is raised to a high value in the teacher model, e.g., 20. Since the soft target of the teacher model is the mean result of multiple components of the teacher model, the training instances are more informative. Therefore, the student model could be trained on much less data than the teacher model. The authors prove the effectiveness on MNIST and speech recognition tasks. Sau *et al.* propose to supervise the training of the student model with multiple teacher models, with the consideration that the distilled knowledge from a single teacher may be limited [262]. They also introduce a noise-based regulariser to improve the health in the performance of the student model.

Romero *et al.* propose FitNet, which extends [259] to create a deeper and lighter student model [260]. Deeper models could better characterise the essence of the data. Both the output of the teacher model and the intermediate representations are used as hints to speed up training of the student model, as well as improve its performance. Opposite to [260], Zagoruyko *et al.* prove that shallow neural networks could also significantly improve the performance of a student model by properly defining attention [261]. Attention is considered as a set of spatial maps that the network focuses the most on in the input to decide the output decision. These maps could be represented as convolutional layers in the network. In the teacher-student paradigm, the spatial attention maps are used to supervise the student model, as shown in Fig. 23.

There are also some efforts focusing on how to design the student model. Crowley *et al.* propose to obtain the student model through replacing the convolutional layers of the teacher model with cheaper alternatives [263]. The new generated student model is then trained under the supervision of the teacher model. Li *et al.* design a framework, named DeepRebirth to merge the consecutive layers without weights, such as pooling and normalisation and convolutional layers vertically or horizontally to compress the model [264]. The newly generated student model learns parameters through layer-wise fine-tuning to minimise the accuracy loss. Fig. 24 presents the framework of DeepRebirth. After compression, GoogLeNet achieves 3x acceleration and 2.5x reduction in runtime memory.

The teacher model is pre-trained in most relevant works. Nevertheless, the teacher model and the student model could be trained in parallel to save time. Zhou *et al.* propose a compression scheme, named Rocket Launching to exploit the

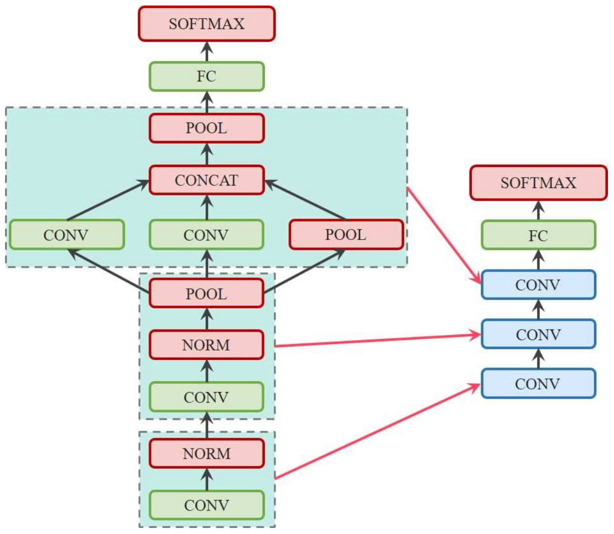


Fig. 24. The illustration of DeepRebirth. The upper model is the teacher model, while the lower is the student model. The highly correlated convolutional layer and non-convolutional layer are merged and become the new convolutional layer of the student model.

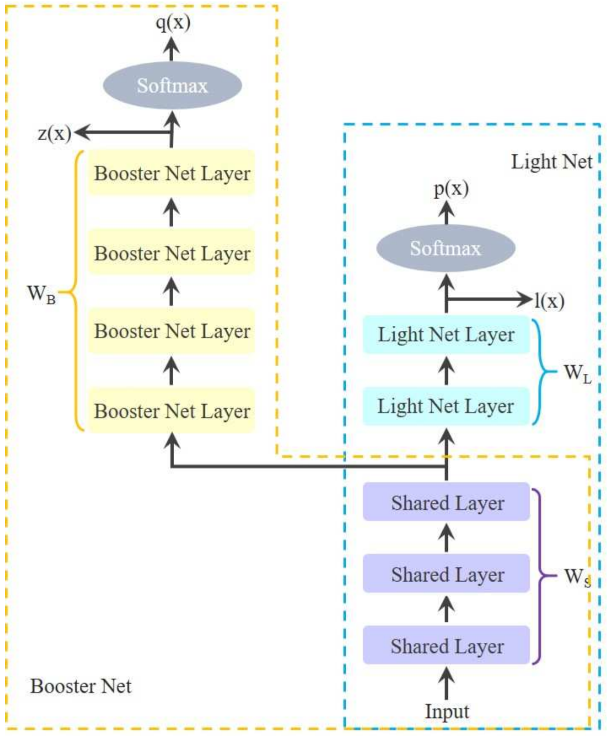


Fig. 25. The structure of Rocket Launching. $\boldsymbol{W_S}$, $\boldsymbol{W_L}$, and $\boldsymbol{W_B}$ denotes parameters. $z(x)$ and $l(x)$ represent the weighted sum before the softmax activation. $p(x)$ and $q(x)$ are outputs. Yellow layers are shared by the teacher and student.

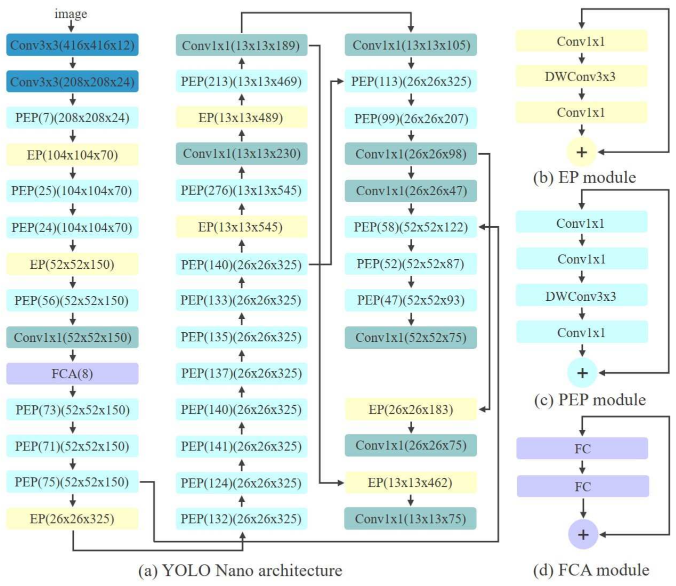


Fig. 26. The architecture of the YOLO Nano network. PEP(x) refers to x channels in PEP, while FCA(x) represents the reduction ratio of x.

simultaneous training of the teacher and student model [265]. During the training, the student model keeps acquiring knowledge learnt by the teacher model through the optimisation of the hint loss. The student model learns both the difference between its output and its target, and the possible path towards the final target learnt by the teacher model. Fig. 25 presents the structure of this framework.

When the teacher model is trained on a dataset concerning with privacy or safety, it is then difficult to train the student model. Lopes *et al.* propose an approach to distill the learned knowledge of the teacher model without accessing the original dataset, which only needs some extra metadata [266]. They first reconstruct the original dataset with the metadata of the teacher model. This step could find the images that best match these given by the network. Then they remove the noise of the image to approximate the activation records through gradients, which could partially reconstruct the original training set of the teacher model.

3) Compact layer design: In deep neural networks, if weights end up to be close to 0, the computation is wasted. A fundamental way to solve this problem is to design compact layers in neural networks, which could effectively reduce the consumption of resources, i.e., memories and computation power. Christian *et al.* propose to introduce sparsity and replace the fully connected layers in GoogLeNet [267]. Residual-Net replaces the fully connected layers with global average pooling to reduce the resource requirements [268]. Both GoogLeNet and Residual-Net achieve the best performance on multiple benchmarks.

Alex *et al.* propose a compact and lightweight CNN model, named YOLO Nano for image recognition [269]. YOLO Nano is a highly customised model with module-level macro- and micro-architecture. Fig. 26 shows the network architecture of YOLO Nano. There are three modules in YOLO Nano: expansion-projection (EP) macro-architecture, residual projection-expansion-projection (PEP) macro-architecture, and a fully-connected attention (FCA) module. PEP could reduce the architectural and computational complexity whilst preserving model expressiveness. FCA enables better utilisation of available network capacity.

Replacing a big convolution with multiple compact layers could effectively reduce the number of parameters and further reduce computations. Iandola *et al.* propose to compress CNN models with three strategies [270]. First, decomposing 3×3 convolution into 1×1 convolutions, since it has much fewer parameters. Second, cut down input channels in 3×3 convolutions. Third, downsample late to produce big feature maps. The larger feature maps could lead to higher classification accuracy. The first two strategies are used to decrease the quantity of parameters in CNN models and the last one is used to maximise the accuracy of the model. Based on three above mentioned strategies, the authors design SqueezeNet, which can achieve $50\times$ reduction in the number of parameters, whilst remaining the same accuracy as the complete AlexNet. Similar approaches ares also used in [271]. Shafiee *et al.* modify SqueezeNet for applications with fewer target classes and they propose SqueezeNet v1.1, which could be deployed on edge devices [272]. Yang *et al.* propose to decompose a recognition task into two simple sub-tasks: context recognition and target recognition, and further design a compact model, namely cDeepArch [273]. This approach uses storage resource to trade for computing resources.

Shen *et al.* introduce Compressive Sensing (CS) to jointly modify the input layer and reduce nodes of each layer for CNN models [275]. CS [512] could be used to reduce the dimensionality of the original signal while preserving most of its information. The authors use CS to jointly reduce the dimensions of the input layer whilst extracting most features. The compressed input layer also enables the reduction of the number of parameters.

Besides the above-mentioned works about CNNs, Zhang *et al.* propose a dynamically hierarchy revolution (DirNet) to compress RNNs [274]. In particular, they mine dictionary atoms from original networks to adjust the compression rate with the consideration of different redundancy degrees amongst layers. They then adaptively change the sparsity across the hierarchical layers.

4) Network pruning: The main idea of network pruning is to delete unimportant parameters, since not all parameters are important in highly precise deep neural networks. Consequently, connections with less weights are removed, which converts a dense network into a sparse one, as shown in Fig. 27 There are some works which attempt to compress neural networks by network pruning.

The work [513] and [514] have taken the earliest steps towards network pruning. They prune neural networks to eliminate unimportant connections by using Hessian loss function. Experiment results prove the efficiency of prunning methods. Subsequent research focuses on how to prune the networks. Han *et al.* propose to prune networks based on a weight threshold [277]. Practically, they first train a model to learn the weights of each connection. The connections with lower weights than the threshold would then be removed. Afterwards, the network is retrained. The pruning approach is straightforward and simple. A similar approach is also used in [278]. In [278], the authors select and delete neurons of low performance, and then use a width multiplier to expand all layer sizes, which could allocate more resources to neurons of high performance. However, the assumption that connections with lower weights contribute less to the results may destroy the structure of the networks.

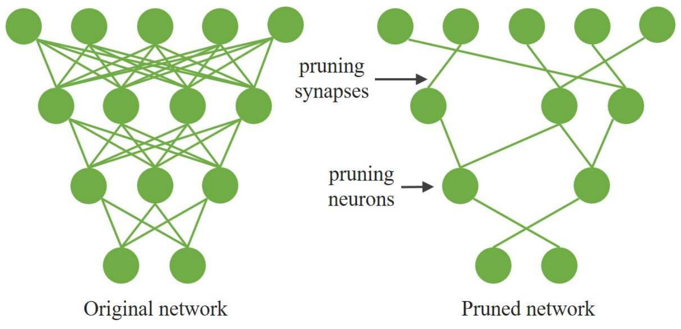


Fig. 27. Illustration of network pruning. Unimportant synapses and neurons would be deleted to generate a sparse network.

Identifying an appropriate threshold to prune neural networks usually takes iteratively trained networks, which consumes a lot of resources and time. Moreover, the threshold is shared by all the layers. Consequently, the pruned configuration maybe not the optimal, comparing with the case of identify thresholds for each layer. To break through these limitations, Manessi *et al.* propose a differentiability-based pruning method to jointly optimise the weights and thresholds for each layer [279]. Specifically, the authors propose a set of differentiable pruning functions and a new regulariser. Pruning could be performed during the back propagation phase, which could effectively reduce the training time.

Molchanov *et al.* propose a new criterion based on the Taylor expansion to identify unimportant neutrons in convolutional layers [280]. Specifically, they use the change of cost function to evaluate the result of pruning. They formulate pruning as an optimisation problem, trying to find a weight matrix that minimises the change in cost function. The formulation is approximately converted to its first-degree Taylor polynomial. The gradient and feature map's activation could be easily computed during back-propagation. Therefore, the approach could train the network and prune parameters simultaneously. You *et al.* propose a global filter pruning algorithm, named Gate Decorator, which transforms a CNN module through multiplying its output by the channel-wise scaling factors [281]. If the scaling factor is set to be 0, the corresponding filter would be removed. They also adopt the Taylor expansion to estimate the change of the loss function caused by the changing of the scaling factor. They rank all global filters based on the estimation and prune according to the rank. Compared with [280], [281] does not require special operations or structures.

In addition to minimum weight and cost functions, there are efforts trying to prune with the metric of energy consumption. Yang *et al.* propose an energy-aware pruning algorithm to prune CNNs with the goal of minimising the energy consumption [282]. The authors model the relationship between data sparsity and bit-width reduction through extrapolating the detailed value of consumed energy from hardware measurements. The pruning algorithm identifies the parts of a CNN that consumes the most energy and prunes the weights to

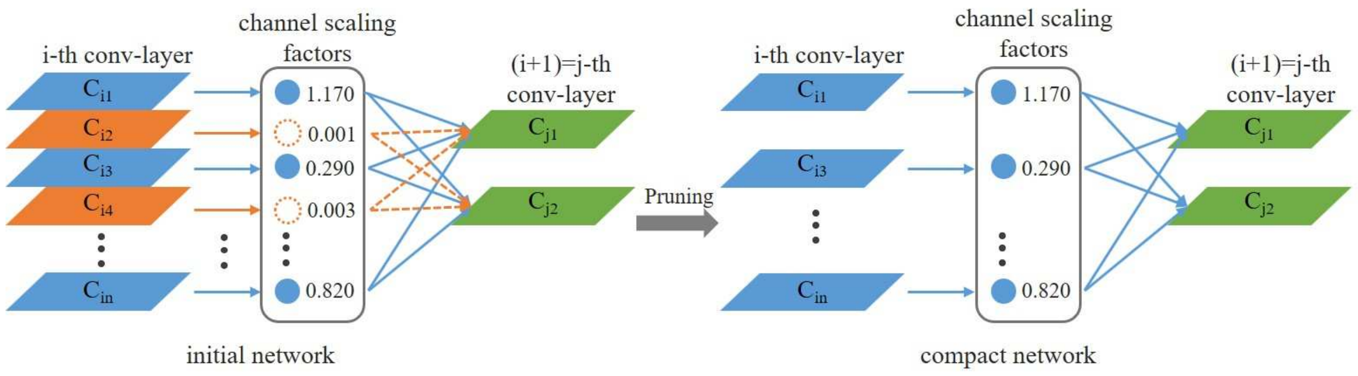


Fig. 28. Each channel is associated with a scaling factor γ in convolutional layers. Then the network is trained to jointly learn weights and scaling factors. After that, the channels with small scaling factors (in orange colour) are pruned, which results in a compact model.

maximise energy reduction.

Yao *et al.* propose to minimise the number of non-redundant hidden elements in each layer whilst retaining the accuracy in sensing applications and propose DeepIoT [283]. In DeepIoT, the authors compress neural networks through removing hidden elements. This regularisation approach is called dropout. Each hidden element is dropouted with a probability. The dropout probability is initialised with 0.5 for all hidden elements. DeepIoT develops a compressor neural network to learn the optimal dropout probabilities of all elements.

Liu *et al.* propose to identify important channels in CNN and remove unimportant channels to compress networks [284]. Specifically, they introduce a scaling factor γ for each channel. The output $\hat{z}$ (also the input of the next layer) could be formulated as $\hat{z} = \gamma z + \beta$, where z is the input of the current layer and β is min-batch. Afterwards, they jointly train the network weight and scaling factors, with L1 regulation imposed on the latter. Following that, they prune the channels with the small scaling factor γ. Finally, the model is fine-tuned, which achieves a comparable performance with the full network. Fig. 28 presents this slimming process. However, the threshold of the scaling factor is not computed, which requires iterative evaluations to obtain a proper one.

Based on network pruning, the work in [285] investigates the data flow inside computing blocks and develops a data reuse scheme to alleviate the bandwidth burden in convolution layers. The data flow of a convolution layer is regular. If the common data could be reused, it is not necessary to load all data to a new computing block. The data reuse is used to parallelise computing threads and accelerate the inference of a CNN model.

5) Parameter quantisation: A very deep neural network usually involves many layers with millions of parameters, which consumes a large amount of storage and slows down the training procedure. However, highly precise parameters in neural networks are not always necessary in achieving high performance, especially when these highly precise parameters are redundant. It has been proved that only a small number of parameters are enough to reconstruct a complete network [510]. In [510], the authors find that the parameters within one layer could be predicted by 5% of parameters, which means we could compress the model by eliminating redundant parameters. There are some works exploiting parameter quantisation for model compression.

Gong *et al.* propose to use vector quantisation methods to reduce parameters in CNN [291]. Vector quantisation is often used in lossy data compression, which is based on block coding [228]. The main idea of vector quantisation is to divide a set of points into groups, which are represented by their central points. Hence, these points could be denoted with fewer coding bits, which is the basis of compression. In [291], the authors use k-means to cluster parameters and quantise these clusters. They find that this method could achieve $16-24\times$ compression rate of the parameters with the scarification of no more than 1% of the top-5 accuracy. In addition to k-means, hash method has been utilised in parameter quantisation. In [292], Chen *et al.* propose to use hash functions to cluster connections into different hash buckets uniformly. Connections in the same hash bucket share the same weight. Han *et al.* combine parameter quantisation and pruning to further compress the neural network without compromising the accuracy [293]. Specifically, they first prune the neural network through recognising the important connections through all connections. Unimportant connections are ignored to minimise computation. Then, they quantise the parameters, to save the storage of parameters. After these two steps, the model will be retrained. These remaining connections and parameters could be properly adjusted. Finally, they use Huffman coding to further compress the model. Huffman coding is a prefix coding, which effectively reduces the required storage of data [515]. Fig. 29 presents the three-step compression.

For most CNNs, the fully connected layers consume most storage in neural network. Compressing parameters of fully connected layers could effectively reduce the model size. The convolutional layers consume most of the times during training and inference. Wu *et al.* design Q-CNN to quantise both fully connected layers and convolutional layers to jointly compress and accelerate the neural network [294]. Similar to [291], the authors utilise k-means to optimally cluster parameters in fully connected and convolutional layers. Then, they quantise parameters by minimising the estimated error of response for each layer. They also propose a training scheme to suppress

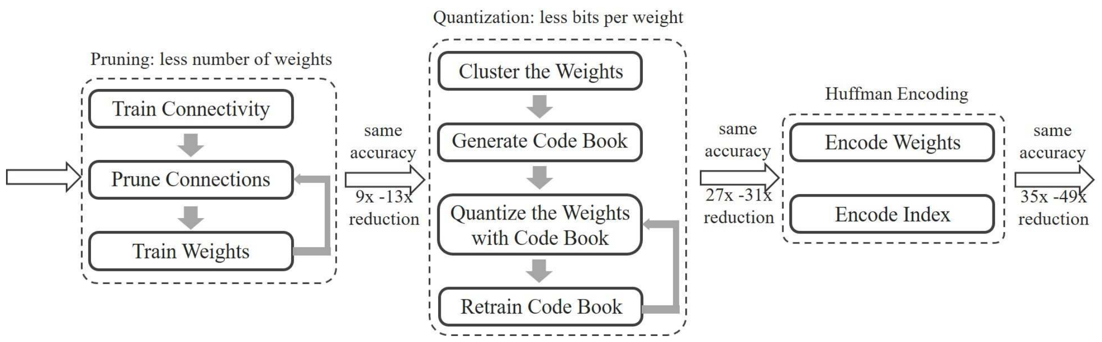


Fig. 29. Illustration of three-stage compression pipeline. First use pruning to reduce the number of weights by $10\times$, then use quantisation to further compress by $27\times$ and $31\times$. Finally use Huffman coding to get more compression.

the accumulative error for the quantisation of multiple layers.

Enormous amount of floating point multiplications consumes significant times and computing resources in inference. There are two potential solutions to address this challenge. The first one is to replace floating point with fixed point, and the second one is to reduce the amount of floating point multiplications.

According to the evaluation of Xilinx, fixed point could achieve the same accuracy results as float [516]. Vanhoucke *et al.* evaluate the implementation of fixed point of an 8-bit integer on x86 platform [299]. Specifically, activation and the weights of intermediate layer are quantised into an 8-bit fixed point with the exception of biases that are encoded as 32-bit. The input layer remains floating point to accommodate possible large inputs. Through the quantisation, the total required memory shrinks $3-4\times$. Results show that the quantised model could achieve a 10x speedup over an optimised baseline and a $4\times$ speedup over an aggressively optimised float point baseline without affecting the accuracy. Similarly, Nasution *et al.* convert floating point to 8 and 16 bits to represent weights and outputs of layers, which lowers the storage to $4.16\times$ [301]. Peng *et al.* quantise an image classification CNN model into an 8-bit fixed-point at the cost of 1% accuracy drop [302]. Anwar *et al.* propose to use L2 error minimisation to quantise parameters [303]. They quantise each layer one by one to induce sparsity and retrain the network with the quantised parameters. This approach is evaluated with MNIST and CIRAR-10 dataset. The results shows that the approach could reduce the required memory by 1/10.

In addition to fixed point, posit number could also be utilised to replace floating point numbers to compress neural networks. Posit number is a unique non-linear numerical system, which could represent all numbers in a dynamic range [517]. The posit number system represents numbers with fewer bits. Float point numbers could be converted into the posit number format to save storage. To learn more about the conversion, readers may refer to [518]. Langroudi *et al.* propose to use the posit number system to compress CNNs with non-uniform data [304]. The weights are converted into posit number format during the reading and writing operations in memory. During the training or inference, when computing operations are required, the number would be converted back to float point. Because this approach only converts the weight between two number systems, no quantisation occurs. The network does not require to be re-trained.

Network Binarisation is an extreme case of weight quantisation. Weight quantisation indicates that all weights are represented by two possible values (e.g., -1 or 1), which could overwhelmingly compress neural networks [295]. For example, the original network requires 32 bits to store one parameter, while in binary connect based network, only 1 bit is enough, which significantly reduces the model size. Another advantage of binary connect is that replacing multiply-accumulate operations by simple accumulations, which could drastically reduce computation in training. Courbariaux *et al.* extend the work [295] further and proposes Binary Neural Network (BNN), which completely changes the computing style of traditional neural networks [296]. Not only the weights, but also the input of each layer is binarised. Hence, during the training, all multiplication operations are replaced by accumulation operations, which drastically improves the power-efficiency. However, substantial experiments indicate that BNN could only achieve good performance on small scale datasets.

Rastegari *et al.* propose a XNOR-net to reduce storage and improve training efficiency, which is different with [296] in the binarisation method and network structure [297]. In Binary-Weight network, all weight values are approximately binarized, e.g., -1 or 1, which reduces the size of network by $32\times$. Convolutions could be finished with only addition and subtraction, which is different with [296]. Hence, the training is speed up $2\times$. With XNOR-net, in addition to weights, the input to convolutional layers are approximately binarised. Moreover, they further simplify the convolution with XNOR operations, which achieves a speed up of $58\times$. The comparison amongst standard convolution, Binary-Weight and XNOR-net is presented as Table. VII.

Lin *et al.* propose to use binary connect to reduce multiplications in DNN [298]. In the forward pass, the authors stochastically binarise weights by binary connect. Afterwards, they quantise the representations at each layer to replace the remaining multiply operations into bit-shifts. Their results show that there is no loss in accuracy in training and sometimes this approach surprisingly achieves even better performance than

TABLE VII
THE COMPARISON AMONGST STANDARD CONVOLUTION, BINARY-WEIGHT AND XNOR-NET.

	Input	**Weight**	**Convolution operation**	**Memory saving**	**Computation saving**	**Accuracy (imageNet)**
Standard Convolution	Real value	Real value	$\times, +, -$	$1\times$	$1\times$	56.7%
Binary-Weight	Real value	Binary value	$+, -$	$\sim 32\times$	$\sim 2\times$	56.8%
XNOR-Net	Binary value	Binary value	XNOR, bitcount	$\sim 32\times$	$\sim 58\times$	44.2%

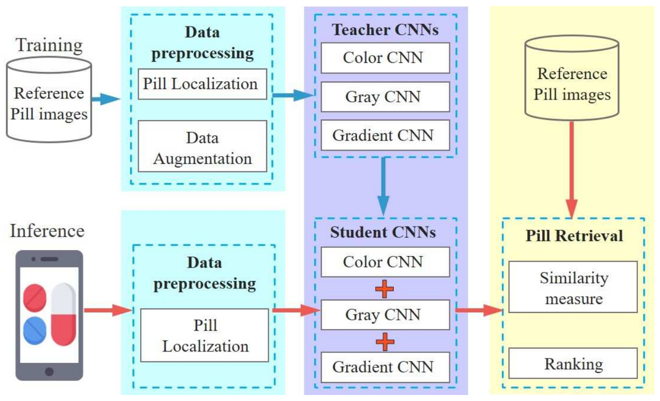


Fig. 30. The architecture of MobileDeepPill. The blue arrows indicates the flow of the training phase, whilst the red arrows indicate the inference phase.

standard stochastic gradient descent training.

Soudry *et al.* prove that binary weights and activations could be used in Expectation Backprogagation (EBP) and achieves high performance [305]. This is based on a variational Bayesian approach. The authors test eight binary text classification tasks with EBP-trained multilayer neural networks (MNN). The results show that binary weights always achieve better performance than continuous weights. Esser *et al.* further develop a fully binary network with the same approach to EBP to improve the energy efficiency on neuromorphic chips [306]. They perform the experimentation on the MNIST dataset, and the results show that the method achieves 99.42% accuracy at 108 μJ per image.

6) Applications: Some efforts try to use these compression techniques on practical applications and prototypes at the edge, including image analysis [307]–[309], automotive [310], [311], and anomaly detection [312], [313].

Mathur *et al.* develop a wearable camera, called DeepEye, that runs multiple cloud-scale deep learning models at edge provide real-time analysis on the captured images [307]. DeepEye enables the creation of five state-of-the-art image recognition models. After camera captures an image, the image pre-processing component deals with the image according to the adopted deep model. There is a model compression component inside the inference engine, which applies available compression techniques to reduce energy consumption and the running time. Finally, DeepEye use the optimised BLAS library to optimise the numeric operations on hardware.

To correctly identify prescription pills for patients based on their visual appearance, Zeng *et al.* develop MobileDeepPill, a pill image recognition system [308]. The pill image recognition model is based on ImageNet [471]. Fig. 30 presents the architecture of MobileDeepPill. In the training phase, the system first localises and splits the pill image in consumer and pill references. The system then enrich samples through running data augmentation module. Finally, the system imports CNNs as the teacher model to supervise the student model. In the inference phase, the system first processes the pill photo and extracts features to perform the student CNNs. As a last step, the system ranks the results according to their possibilities.

Wang *et al.* propose a fast image search framework to implement the content-based image retrieval (CBIR) service from cloud servers to edge devices [309]. Traditional CBIR services are based on the cloud, which suffers from high latency and privacy concerns. The authors propose to reduce the resource requirements of the model and to deploy it on edge devices. For the two components consuming most resources, i.e., object detection and feature extraction, the authors use low-rank approximation to compress these two parts. The compressed model achieves 6.1$\times$ speedup for inference.

With growing interests from the automotive industry, various large deep learning models with high accuracy have been implemented in smart vehicles with the assistance of compression techniques. Kim *et al.* develop a DL based object recognition system to recognise vehicles [310]. The vehicle recognition system is based on faster-RCNN. To deploy the system on vehicles, the authors apply network pruning and parameter quantisation to compress the network. Evaluations show that these two compression techniques reduce the network size to 16% and reduce runtime to 64%. Xu *et al.* propose an RNN based driving behaviour analysis system on vehicles [311]. The system uses the raw data collected by a variety of sensors on vehicles to predict the driving patterns. To deploy the system on automobiles, the authors apply parameter quantisation to reduce the energy consumption and model size. After compression, the system size is reduced to 44 KB and the power overhead is 7.7 mW.

In winter, electric transmission lines are susceptible to freezes, which may break these lines and result in great loss in power system. Applying image recognition algorithms to measure ice thickness is promising. However, current ordinary image recognition models are computation-intensive, which makes it impossible to be executed on monitoring terminals. Wang *et al.* design a lightweight recognition model to solve the problem [312]. Specifically, they first use MobileNet-v3 [519] to extract features and SSD [520] to extract high-dimensional features. Then they apply network pruning to compress the model. Experiments show that the lightweight algorithm achieves 74.5% accuracy. Shang *et al.* use network pruning and parameter quantization to compress an

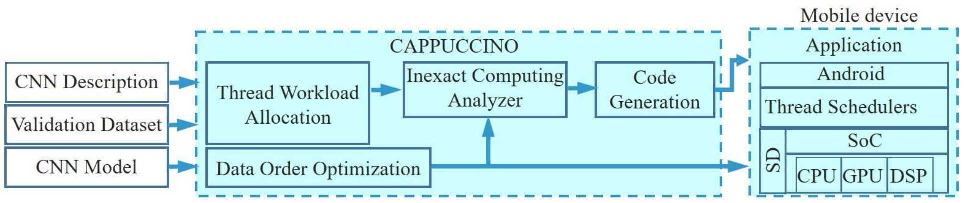


Fig. 31. The architecture of Cappuccino. Thread workload allocation component optimises the workload of each thread. Data order optimisation component converts data format. Inexact computing analyser determines the tradeoff amongst multiple metrics.

anomaly detection algorithm on electrical equipments [313]. The anomaly detection algorithm is based on AlexNet, which judge the status of electrical equipments through equipment images. After compression, the lightweight detection algorithm can be deployed on edge devices.

C. Inference Acceleration

The computing capacities of edge devices have been increased and some embedded devices, such as NVIDIA Jetson TX2 [521] could directly perform CNN. However, it is still difficult for most edge devices to directly run large models. Model compression techniques reduce the required resources to create neural network models and facilitate the performance of these models on edge devices. Model acceleration techniques further speed up the performance of the compressed model on edge devices. The main idea of model acceleration in inference is to reduce the run-time of inference on edge devices and realise real-time responses for specific neural network based applications without changing the structure of the trained model. According to acceleration approaches, research works on inference acceleration could be divided into two categories: hardware acceleration and software acceleration. Hardware acceleration methods focuses on parallelising inference tasks to available hardware, such as CPU, GPU, and DSP. Software acceleration method focuses on optimising resource management, pipeline design, and compiler.

1) Hardware Acceleration: Recently, mobile devices are becoming increasingly powerful. More and more mobile platforms are equipped with GPUs. Since mobile CPUs are not suitable for the computing of deep neural networks, the embedded GPU could be used to share the computing tasks and accelerate the inference. Table VIII summaries existing literature on hardware acceleration.

Alzantot *et al.* evaluate the performance of CNNs and RNNs only on CPU, and compares against the execution in parallel on all available computing resources, e.g., CPU, GPU, DSP, etc. [314]. Results show that the parallel computing paradigm is much faster. Loukadakis *et al.* propose two parallel implementations of VGG-16 network on ODROID-XU4 board: OpenMP version and OpenCL version [315]. The former parallelises the inference within the CPU, whilst the latter one parallelises within the Mali GPU. These two approaches achieve $2.8\times$ and $11.6\times$ speedup, respectively. Oskouei *et al.* design a mobile GPU-based accelerator for using deep CNN on mobile platforms, which executes inference in parallel on both CPU and GPU [316]. The accelerator achieves $60\times$ speedup. The authors further develop a GPU-based accelerated library for Android devices, called CNNdroid, which could achieve up to $60\times$ speedup and $130\times$ energy reduction Android platforms [317].

With the consideration that the memory on edge devices are usually not sufficient for neural networks, Tsung *et al.* propose to optimise the flow to accelerate inference [318]. They use a matrix multiplication function to improve the cache hit rate in memory, which indirectly speeds up the execution of the model.

Nvidia has developed a parallelisation framework, named Compute Unified Device Architecture (CUDA) for desktop GPUs to reduce the complexity of neural networks and improve inference speed. For example, in [522], CUDA significantly improves the execution efficiency of RNN on desktop GPUs. Some efforts implement the CUDA framework onto mobile platforms. Rizvi *et al.* propose an approach for image classification on embedded devices based on the CUDA framework [319]. The approach features the most common layers in CNN models, e.g., convolutions, max-pooling, batch-normalisation, and activation functions. General Purpose Computing GPU (GPGPU) is used to speed up the most computation-intensive operations in each layer. The approach is also used to implement an Italian license plate detection and recognition system on tablets [320]. They subsequently introduce matrix multiplication to reduce the computational complexity of convolution in a similar system to achieve real-time object classification on mobile devices [321]. They also apply the approach in a robotic controller system [322].

However, the experiments in [323] show that directly applying CUDA on mobile GPUs may be ineffective, or even deteriorates the performance. Cao *et al.* propose to accelerate RNN on mobile devices based on a parallelisation framework, called RenderScript [323]. RenderScript [324] is a component of the Android platform, which provides an API for hardware acceleration. RenderScript could automatically parallelise the data structure across available GPU cores. The proposed framework could reduce latency by $4\times$.

Motamedi *et al.* implement SqueezeNet on mobile and evaluates the performance on three different Android devices based on RenderScript [325]. Results show that it achieves $310.74\times$ speedup on a Nexus 5. They further develop a general framework, called Cappuccino, for automatic synthesis of efficient inference on edge devices [326]. The structure of Cappuccino is shown as in Fig. 31. There are three inputs for the framework: basic information of the model, model file, and dataset. There are three kinds of parallelisation: kernel-level,

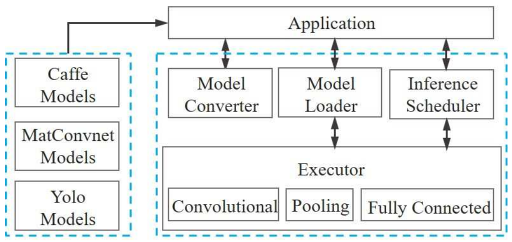


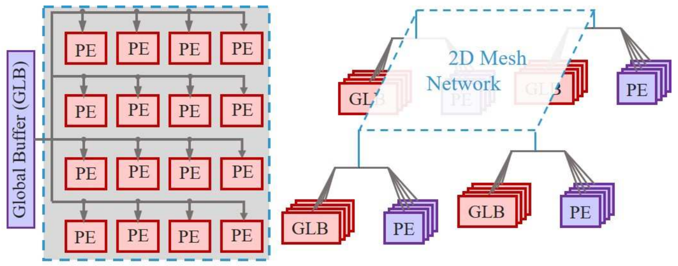
Fig. 32. The structure of Deepsense. The model converter converts the format of the input model, then the model loader loads the model into memory. Inference scheduler is responsible for task scheduling for GPU. The executor runs the allocated tasks on a GPU.

filter bank-level, and output-level parallelisation. The thread workload allocation component allocates tasks by using these three kinds of parallelisation. They specially investigate the optimal degree of concurrency for each task, i.e., the number of threads in [327]. The data order optimisation component is used to convert the data format. Cappuccino enables imprecise computing in exchange for high speed and energy efficiency. The inexact computing analyser component is used to analyse the effect of imprecise computing and determine the tradeoff amongst accuracy, speed and energy efficiency.

Huynh *et al.* propose Deepsense, a GPU-based CNN framework to run various CNN models in soft real-time on mobile devices with GPUs [328]. To minimise the latency, Deepsense applies various optimisation strategies, including branch divergence elimination and memory vectorisation. The structure of Deepsense is shown as in Fig. 32. The model converter first converts pre-trained models with different representations into a pre-defined format. Then, the model loader component loads the converted model into memory. When inference starts, the inference scheduler allocates tasks to the GPU sequentially. The executor takes inputted data and the model for executing. During the execution pipeline, CPU is only responsible for padding and intermediate memory allocation, whilst most computing tasks are done by the GPU. The authors further present a demo of the framework in [118] for continuous vision sensing applications on mobile devices.

The heterogeneous multi-core architecture, including CPU and GPU on mobile enables the application of neural networks. By reasonably mapping tasks to cores could improve energy efficiency and inference speed. Taylor *et al.* propose a machine learning based approach to map OpenCL kernels onto proper heterogeneous multi-cores to achieve given objectives, e.g., speedup, energy-efficiency or a tradeoff [329]. The framework first trains the mapping model with the optimisation setting for each objective, then it uses the learned model to schedule OpenCL kernels based on the information of the application.

Rallapalli *et al.* find that the memory of GPUs severely limits the operation of deep CNNs on mobile devices, and proposes to properly allocate part of computation of the fully-connected layers to the CPU [330]. The fully-connected layers are split into several parts, which are executed sequentially. Meanwhile, part of these tasks are loaded into the memory of the CPU for processing. They evaluate the method with an object detection model, YOLO [331] on Jetson TK1 board and achieve 60× speedup.



(a) Original Eyeriss (b) Eyeriss v2

Fig. 33. The comparison between Eyeriss and Eyeriss v2. Both of them are composed of GLB and PE. Eyeriss v2 adopts a hierarchical structure to reduce communication cost.

In addition to commonly used hardware, i.e., CPUs, mobile GPUs, GPGPU, and DSP, field-programmable gate arrays (FPGAs) could also be used for acceleration. Different from CPUs and GPUs, which run software code, FPGA uses hardware level programming, which means that FPGA is much faster than CPU and GPU. Bettoni *et al.* implement an object recognition CNN model on FPGA via Tiling and Pipelining parallelisation [332]. Ma *et al.* exploit the data reuse and data movement in a convolution loop and proposes to use loop optimisation (including loop unrolling, tiling, and interchange) to accelerate the inference of CNN models in FPGA [333]. A similar approach is also adopted in [334].

Lots of literature focus on developing energy-efficiency DNNs. However, the diversity of DNNs makes them inflexible for hardware [335]. Hence, some researchers attempt to design special accelerating chips to flexibly use DNNs. Chen *et al.* develop an energy-efficient hardware accelerator, called Eyeriss [336]. Eyeriss uses two methods to accelerate the performance of DNNs. The first method is to exploit data reuse to minimise data movement, whilst the second method is to exploit data statistics to avoid unnecessary reads and computations, which improves energy efficiency. Subsequently, they change the structure of the accelerator and propose a new version, Eyeriss v2, to run compact and sparse DNNs [337]. Fig. 33 shows the comparison between Eyeriss and Eyeriss v2. Both of them consist of an array of processing elements (PE) and global buffers (GLB). The main difference is the structure. Eyeriss v2 is hierarchical, in which PEs and GLBs are grouped to reduce communication cost.

2) Software Acceleration: Different from hardware acceleration, which depends on the parallelisation of tasks on available hardware, software acceleration mainly focuses on optimising resource management, pipeline design, and compilers. Hardware acceleration methods speed up inference through increasing available computing powers, which usually does not affect the accuracy, whilst software acceleration methods maximise the performance of limited resources for speedup, which may lead to a drop in accuracy with some cases. For example, in [339], the authors sacrifice accuracy for real-time response. Table IX summarises existing literature on software acceleration.

Georgiev *et al.* investigate the tradeoff between performance

TABLE VIII
LITERATURE SUMMARY OF HARDWARE ACCELERATION.

Ref.	Model	Executor	Enabler	Object	Baseline	Performance
[314]	CNN, RNN	CPU, GPU	RenderScript	Feasibility check	Nexus 5x	3× faster
[315]	VGG-16	CPU, GPU	SIMD	Speed up inference	Initial OpenCL	11.6× faster
[316]	CNN	GPU	SIMD	Speed up inference	CPU-only LeNet5	60× faster
[317]	CNN	GPU	SIMD	Speed up inference	CPU-only LeNet	60× faster 130× energy-saving
[318]	DNN	GPU	Flow optimisation	Enable DNN on mobile device	Pure CPU DNN	58× faster 104× energy-saving
[319]	CNN	GPGPU	CUDA	Maximise throughput	CPU-only AlexNet	50× faster
[320]	DNN	GPU	CUDA	Real-time character detection	Complete NN	250ms per time
[321]	DNN	GPU	Matrix multiplication	Real-time character detection	ResNet-34	3× faster
[323]	LSTM	GPU	RenderScript	Rnn RNN on mobile platform	CPU-only LSTM	4× faster
[325]	CNN	GPU	RenderScript	Acceleration, energy-efficiency	Sequential SqueezeNet	310.74× faster 249.47× less energy
[327]	CNN	CPU, GPU, DSP	RenderScript	Optimise thread number	GoogLeNet	2.37× faster
[326]	CNN	CPU, GPU, DSP	RenderScript	Automatic speedup	SqueezeNet	272.03× faster
[328]	CNN	GPU	Memory vectorisation	Real-time response	VGG-F	361ms runtime
[118]	VGG16	GPU	Tucker decomposition	Real-time response	Pure VGG16	644ms runtime
[329]	SVM	CPU, GPU	Kernel mapping	Adaptive optimisation	OPENCL scheme	1.2× faster 1.6× energy saving
[330]	YOLO	CPU, GPU	Memory optimisation	Enable CNN on mobile device	Deep CNN	0.42s for YOLO
[332]	CNN	FPGA	Tiling, Pipelining	Enable CNN in FPGA	CNN on SoC GPU	15× faster
[333]	VGG16	FPGA	Loop optimisation	Memory and data movement	Virtex-7 FPGA	3.2× faster
[334]	CNN	FPGA	Loop optimisation	Improve energy efficiency	CNN on i7 processor	23% faster 9.05× energy-saving
[336]	VGG16	Eyeriss	Data reuse	Improve energy efficiency	Without data gating	45% power saving
[337]	AlexNet	Eyeriss v2	Hierarchical mesh	Processing efficiency	Eyeriss	12.6× faster 2.5× energy-saving
[338]	CNN	TPU	Systolic tensor array	Improve systolic array	Systolic array	3.14× faster

and energy consumption of an audio sensing model on edge devices [340]. Work items need to access different kinds of memory, i.e., global, shared, and private memory. Global memory has the maximum size but minimum speed, whilst private memory is fastest and smallest but exclusive to each work item. Shared memory is between global and private memory. Typical audio sensing models have the maximum parameters, which surpasses the capacity of memory. They use memory access optimisation techniques to speed up the inference, including vectorisation, shared memory sliding window, and tiling.

Lane *et al.* propose DeepX to reduce the resource usage on mobile devices based on two resource management algorithms [341]. The first resource management algorithm is for runtime layer compression. The model compression method discussed in Section VI-B could also be used to remove redundancy from original layers. Specifically, they use a SVD-based layer compression technique to simplify the model. The second algorithm is for architecture decomposition, which decomposes the model into blocks that could be performed in parallel. The workflow of DeepX is shown in Fig. 34. They further develop a prototype of DeepX on wearable devices [342]. Subsequently, they develop the DeepX toolkit (DXTK) [343]. A number of pre-trained and compressed deep neural network models are packaged in DXTK. Users could directly use DXTK for specific applications.

Yang *et al.* propose an adaptive software accelerator, Netadpt, which could dynamically speed up the model according to specific metrics [344]. They use empirical measurements on practical devices to evaluate the performance of the accelerator. Fig. 35 shows the structure of Netadapt. Netadpat adjusts the network according to the given budget, i.e., latency, energy, etc. During iteration, the framework generates many network proposals. Then, Netadapt evaluates these proposals according to direct empirical measurements, and selects one with maximum accuracy. The framework is similar to [523], which caches multiple model compression systems, and compresses the input model according to users' demand.

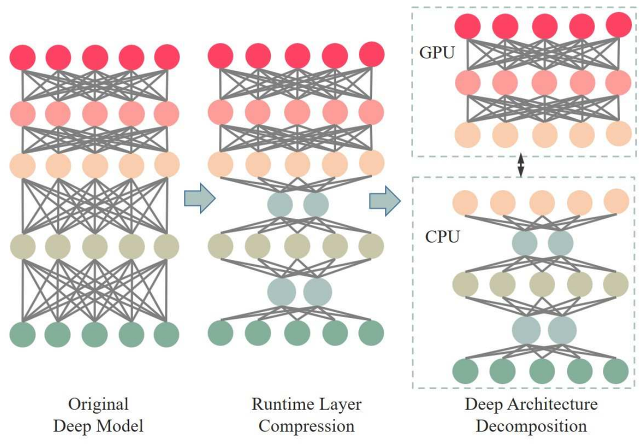


Fig. 34. The workflow of DeepX. Layer compression could reduce the requirement on resource, whilst the architecture decomposition divides the model into multiple blocks that could be performed in parallel.

Ma *et al.* introduce the concept of quality of service (QoS) in model acceleration and develop an accelerator, DeepRT [345]. The QoS of an accelerated model is defined as a tuple

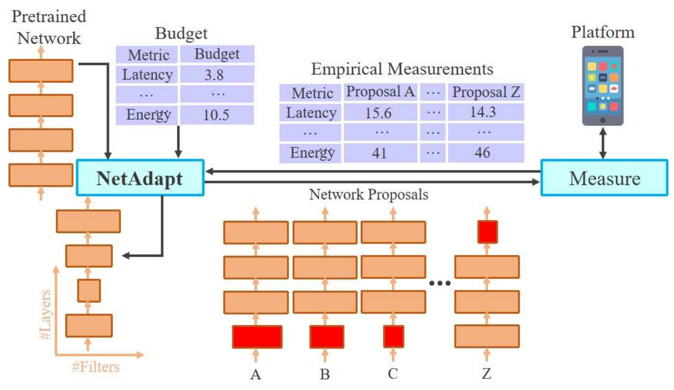


Fig. 35. The structure of Netadapt. Netadapt caches multiple pre-trained models. When requests arrive, Netadapt selects a specific model and adjusts its structure according to the given budget. Then it chooses the best proposal as the accelerating scheme according to empirical measurement.

$Q = (d, C)$, where d is a desired response time and C denotes model compression bound. There is a QoS manager component in DeepRT, which controls the system resources to support the QoS during the acceleration.

Liu *et al.* find that fast Fourier transform (FFT) could effectively speed up convolution operation [524]. Abtahi *et al.* applie FFT-based convolution ResNet-20 on NVIDIA Jetson TX1 and evaluates the performance [346]. Results show the inference speed is improved several times. However, FFT-based convolution only works when the convolution kernel is big, e.g., bigger than $9 \times 9 \times 9$. Most models adopt smaller kernels in practice. Hence, there are few applications of FFT-based convolution in practice.

In continuous mobile vision applications, mobile devices are required to deal with continuous videos or images for classification, object recognition, text translation, etc. These continuous videos or images contain large amounts of repeated frames, which are computed through the model again and again during the inference. In such applications, caching mechanisms are promising for acceleration. Xu *et al.* propose CNNCache, a cache-based software accelerator for mobile continuous vision applications, which reuses the computation of similar image regions to avoid unnecessary computation and saves resources on mobile devices [46]. Cavigelli *et al.* present a similar framework, named CBinfer [119]. The difference is that CBinfer considers the threshold of the pixel size when matching frames. However, CBinfer only matches frames of the same position, which may be ineffective in mobile scenarios. [118] also considers reusing the result of the similar input in inference. Different from [46] and [119], the authors extract histogram-based features to match frames, instead of comparing pixels.

VII. Edge Offloading

Computation is of utmost importance for supporting edge intelligence, which powers the other three components. Most edge devices and edge servers are not as powerful as central servers or computing clusters. Hence, there are two approaches to enable computation-intensive intelligent applications at the edge: reducing the computational complexity of applications and improving the computing power of edge devices and edge servers. The former approach has been discussed in previous sections. In this section, we focus on the latter approach.

Considering the hardware limitation of edge devices, computation offloading [18], [466], [527]–[529] offers promising approaches to increase computation capability. Literature of this area mainly focuses on designing an optimal offloading strategy to achieve a particular objective, such as latency minimisation, energy-efficiency, and privacy preservation. According to their realisation approaches, these strategies could be divided into five categories: device-to-cloud (D2C), device-to-edge (D2E), device-to-device (D2D), and hybrid architecture.

A. D2C offloading strategy

It consumes considerable computing resources and energy to deal with streamed AI tasks, such as video analysis and continuous speech translation. Most applications, such as Apple Siri and Google Assistant, adopt pure cloud based offloading strategy, in which devices upload input data, e.g., speech or image to cloud server through cellular or WiFi networks. The inference through a giant neural model with high accuracy is done by powerful computers and the results are transmitted back through the same way. There are three main disadvantages in this procedure: (1) mobile devices are required to upload enormous volumes of data to the cloud, which has proved to be the bottleneck of the whole procedure [350]. Such a bottleneck increases users' waiting time; (2) the execution depends on the Internet connectivity; once the device is offline, relative applications could not be used; (3) the uploaded data from mobile devices may contain private information of users, e.g., personal photos, which might be attacked by malicious hackers during the inference on cloud server [530]. There are some efforts trying to solve these problems, which will be discussed next. Table X summarises existing literature on D2C offloading strategy.

There are usually many layers in a typical deep neural network, which processes the input data layer by layer. The size of intermediate data could be scaled down through layers. Li *et al.* propose a deep neural network layer schedule scheme for the edge environment, leveraging this characteristic of deep neural networks [347]. Fig. 36 shows the structure of neural network layer scheduling-based offloading scheme. Edge devices lacking computing resources, such as IoT devices, first upload the collected data to nearby edge server, which would process the original input data through few low network layers. The generated intermediate data would be uploaded to the cloud server for further processing and eventually output the classification results. The framework is also adopted in [348]. The authors use edge server to pre-process raw data and extract key features.

The model partitioning and layer scheduling could be designed from multiple perspectives, e.g., energy-efficiency, latency, and privacy. Eshratifar *et al.* propose a layer scheduling algorithm from the perspective of energy-efficiency in a similar offloading framework [349]. In particular, they obtain the optimal layer partitioning scheme by solving the integer

TABLE IX
LITERATURE SUMMARY OF SOFTWARE ACCELERATION.

Ref.	Model	Enabler	Object	Performance & Baseline	Accuracy
[339]	DNN	Memory access optimisation	Performance-energy tradeoff	83% accuracy, VGG	Lossy
[340]	DNN	Resource management	Accelerate inference	3-4× less energy, Gaussian Mixture Models	Lossless
[341]	DNN	Compression, decomposition	Reduce resource use	5.8× faster, Cloud offloading	4.9% loss
[342]	DNN	Compression, decomposition	Reduce resource use	5.8× faster, DeepX	4.9% loss
[344]	NN	Caching	Adaptive speedup	1.7× speedup, MobileNets	4.9% Lossless
[525]	DNN	Caching, model selection	Optimizing DL inference	1.8× speedup, single model strategy	7.52% higher
[346]	ResNet-20	FFT-based convolution	Accelerate convolution	6.8× throughput, Direct-Conv	N/A
[46]	CNN	Cache mechanism	Accelerate inference	20.2% faster, without caching	3.51% drop
[119]	CNN	Caching, pixel matching	Accelerate inference	9.1× faster, without caching	0.1% drop
[118]	YOLO	Caching, feature extraction	Real-time response	36% faster, without caching	3.8%-6.2% drop
[526]	NN	Optimized computing library	Ultra-low-power computing	Up to 63× faster, sequential execution	Lossless

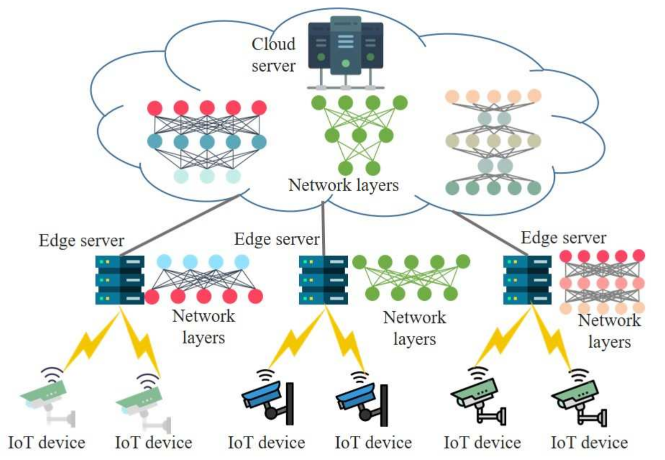


Fig. 36. The structure of neural network layer scheduling-based offloading. IoT devices first upload collected data to edge server, where few neural network layers are deployed. The raw data is first pre-processed on edge servers. Then the intermediate results are uploaded to cloud server for further processing.

linear programming (ILP) problem. Kang *et al.* investigate this problem between edge and cloud side [350]. They propose to partition the computing tasks of DNN between local mobile devices and cloud server and design a system, called Neurosurgeon, to intelligently partition DNN based on the prediction of system dynamics. Osia *et al.* consider the layer scheduling from the perspective of privacy preservation [351]. They add a feature extractor module to identify private features from raw data, which will be sent to the cloud for further processing. Analogous approaches are also adopted in [352], [353].

In continuous computer vision analysis, video streams need to be uploaded to the cloud server, which requires a large amount of network resources and consumes battery energy. Ananthanarayanan *et al.* propose a geographically distributed architecture of clouds and edge servers for real-time video analysis [354]. Fixed (e.g., traffic light) and mobile cameras (e.g., event data recorder) upload video streams to available edge servers for pre-processing. The pre-processed data would be further transmitted to a central cloud server in a geographic location for inference. Similarly, Ali *et al.* leverage all available edge resources to pre-process data for large-scale video stream analytics [355]. Deep learning based video analytic applications contain four stages, including motion detection, frame enhancement, object detection based on shallow networks, and object detection based on deep networks. With the traditional cloud-based approach, these four stages are executed on a cloud server. The authors propose to execute the first two stages locally, which does not require much computation capacity. The output is then transmitted to edge servers for further processing (the third stage). The output is then uploaded to the cloud for final recognition.

The easiest offloading strategy is to offload the inference task to the cloud when the network condition is good, otherwise perform model compression locally. For example, [356] only considers the network condition when offloading healthcare inference tasks. Similar idea is also adopted in [357]–[360], [531].

Georgiev *et al.* consider a collective offloading scheme for heterogeneous mobile processors and cloud for sensor based applications, which makes best possible use out of different kinds of computing resources on mobile devices, e.g., CPU, GPU, and DSP [361]. They designed a task scheduler running on low-power co-processor unit (LPU) to dynamically restructure and allocate tasks from applications across heterogeneous computing resources based on fluctuations in device and network. Specifically, the optimal scheduling is achieved by solving a mixed integer linear programming (MILP) problem.

B. D2E offloading strategy

Three main disadvantages with the D2C offloading strategy have been discussed, i.e., latency, wireless network dependency, and privacy concern. Although various solutions have been proposed to alleviate these problems, they do not address these fundamental challenges. Users still need to wait for a long time. Congested wireless networks lead to failed inference. Moreover, the potential risk of private information leakage still exists. Hence, some works try to explore the potential of D2E offloading, which may effectively address these three problems. Edge server refers to the powerful servers (more powerful than ordinary edge devices) that is physically near mobile devices. For example, wearable devices could offload the inference tasks to their connected smartphones. Smartphones could offload computing tasks to cloudlets deployed at roadside. Table XI summarises the existing literature on D2E offloading strategy.

TABLE X
LITERATURE SUMMARY OF D2C OFFLOADING STRATEGY.

Ref.	Enabler	Platform	Focus and problem	Baseline	Performance
[347]	Online algorithm	Edge and cloud	Layer partition to reduce uploaded data	Fixed mode	Complete more tasks
[348]	Static partition	Edge and cloud	Prototype design	Local running	$3.23\times$ faster
[349]	ILP	Edge and cloud	Layer partition for energy-efficiency	Local running	$3.07\times$ faster
[350]	Runtime prediction	Edge and cloud	Layer partitioning for latency, energy	Cloud-only	$3.1\times$ faster, 140.5% higher
[351]	Feature extraction	Local and cloud	Layer partitioning for privacy	N/A	94% accuracy
[352]	Feature obfuscation	Local and cloud	Privacy protection	DFT, DCT	$8\times$ higher utility
[353]	Feature obfuscation	Local and cloud	Privacy protection	DEEProtect	Smaller false rejection rate
[355]	Static partition	Edge and cloud	Task allocation for QoS	Cloud-only	$3.1\times$ faster, 140.5% higher
[356]	Binary strategy	Edge or cloud	Offloading decision for acceleration	Without offloading	80% energy-saving
[358]	Binary strategy	Edge or cloud	Model schedule, Latency-accuracy tradeoff	Fast R-CNN	78ms vs. 142ms (baseline)
[359]	Binary strategy	Edge or cloud	Multi-objective, real-time response	Local running	Higher accuracy and framerate
[360]	Binary strategy	Edge or cloud	Multi-objective, real-time response	Local-only	Higher accuracy
[361]	MILP	Local and cloud	Optimal schedule for energy efficiency	Cloud offloading	$1.6-3\times$ energy-efficiency

First, we review the works that offload inference tasks to specialised edge servers, e.g., cloudlets and surrogates [532], which refer to infrastructure deployed at edge of the network. There are two general problems that need to be considered in this scenario, including (1) which component of the model could be offloaded to the edge; and (2) which edge server should be selected to offload to. Ra *et al.* develop a framework, named Odessa for interactive perception applications, which enables parallel execution of the inference on local devices and edge server [362]. They propose a greedy algorithm to partition the model based on the interactive deadlines. The edge servers and edge devices in Odessa are assumed to be fixed, meaning they do not consider problem (2). Streiffer *et al.* appoint an edge server for mobile devices that requests video frame analytics [363]. They evaluate the impact of distance between edge server and mobile devices on latency and packet loss and find that offloading inference tasks to an edge server at a city-scale distance could achieve the similar performance with execution locally on each mobile devices.

Similar to D2C offloading, where the partitioned model layers could be simultaneously deployed on both cloud server and local edge device, the partitioned model layers could also be deployed on edge servers and edge devices. This strategy reduces the transmitted data, which further reduces latency and preserve privacy. Li *et al.* propose Edgent, a device-edge co-inference framework to realise this strategy [364]. The core idea of Edgen is to run computation-intensive layers on powerful edge servers and run the rest layers on device. They also adopt model compression techniques to reduce the model size and further reduce the latency. Similarly, Ko *et al.* propose a model partitioning approach with the consideration of energy efficiency [365]. Due to the difference of available resources between edge devices and edge servers, partitioning the network at a deeper layer would reduce the energy efficiency. Hence, they propose to partition the network at the end of the convolution layers. The output features through the layers on edge device would be compressed before transmitted to edge server to minimise the bandwidth usage.

Some efforts [533]–[535] attempt to encrypt the sensitive data locally before uploading. On cloud side, non-linear layers of a model are converted into linear layers, and then they use homomorphic encryption to execute inference over encrypted input data. This offloading paradigm could also be adopted on edge servers. However, the encryption operation is also computation-intensive. Tian *et al.* propose a private CNN inference framework, LEP-CNN, to offload most inference tasks to edge servers and to avoid privacy breaches [366]. The authors propose an online/offline encryption method to speed up the encryption, which trades offline computation and storage for online computation speedup. The execution of the inference over encrypted input data on edge server addresses privacy issues.

Mobility of devices introduces a challenge during the offloading, e.g., in autonomous driving scenarios. Mobile devices may lose the connection with edge server before the inference is done. Hence, selecting an appropriate edge server according to users' mobility pattern is crucial. Zhang *et al.* use reinforcement learning to decide when and which edge server to offload to [367]. A deep Q-network (DQN) based approach is used to automatically learn the optimal offloading scheme from previous experiences. If one mobile device moves away before the edge server finishes the execution of the task, the edge server must drop the task, which wastes the computing resources. Jeong *et al.* propose to move the execution state of the task back to the mobile device from the edge server before the mobile device moves away in the context of web apps [368]. The mobile device continues the execution of the task in this way.

Since the number of edge servers and computation capacity of edge servers are limited, edge devices may compete for resources on edge servers. Hence, proper task scheduling and resource management schemes should be proposed to provide better services at edge. Yi *et al.* propose a latency-aware video edge analytic (LAVEA) system to schedule the tasks from edge devices [369]. For a single edge server, they adopt Johnson's rule [536] to partition the inference task into a two-stage job and prioritise all received offloading requests from edge devices. LAVEA also enables cooperation among different edge servers. They propose three inter-server task scheduling algorithms based on transmission time, scheduling time, and queue length, respectively.

TABLE XI
LITERATURE SUMMARY OF D2E OFFLOADING STRATEGY.

Ref.	Enabler	Problem	Object	Baseline	Performance
[362]	Bottleneck estimation	Model partitioning	Responsiveness and accuracy	Domain experts	3× faster
[363]	SDN	Task scheduling	Responsiveness and accuracy	Local processing	3× faster
[364]	Online algorithm	Model partitioning Network pruning	Reduce latency	Local processing	100-1000ms runtime
[365]	Split convolution layers	Model partitioning	Energy-efficiency	Host inference	4.5× energy-saving
[366]	Online encryption	Privacy protection	Privacy and latency	Without offloading	35× faster, 95.56% energy-saved
[368]	Execution state migration	Mobility factors	Minimize inference time	Client-only strategy	Much faster
[369]	Johnson's rule	Task scheduling	Minimise latency	Local running	$1.2-1.7\times$ faster

C. D2D offloading strategy

Most neural networks could be executed on mobile devices after compression and achieve a compatible performance. For example, the width-halved GoogLeNet on unmanned aerial vehicles achieves 99% accuracy [537]. Some works consider a more static scenario, where edge devices, such as smart watches are linked to smartphones or home gateways. Wearable devices could offload their model inference tasks to connected powerful devices. There are two kinds of offloading paradigms in this scenario, including binary decision-based offloading and partial offloading. Binary decision offloading refers to executing the task either on a local device or through offloading. This paradigm is similar to D2C offloading. Partial offloading means dividing the inference task into multiple sub-tasks and offloading some of them to associated devices. In fact, although the associated devices are more powerful, the performance of the complete offloading is not necessarily better than partial offloading. Because complete offloading is required to transmit complete input data to the associated device, which increases the latency. Table XII summarises the existing literature for D2D offloading strategy.

Xu *et al.* present CoINF, an offloading framework for wearable devices, which offloads partial inference tasks to associated smartphones [370]. CoINF partitions the model into two sub-models, in which the first sub-model could be executed on the wearable devices, while the second sub-model could be performed on smartphones. They find that the performance of partial offloading outperforms the complete offloading in some scenarios. They further develop a library and provide API for developers. Liu *et al.* also propose EdgeEye, an open source edge computing framework to provide real-time service of video analysis, which provides a task-specific API for developers [371]. Such APIs help developers focus on application logic. Similar methods are also adopted in [538].

If one edge device is not powerful enough to provide real-time response for model inference, a cluster of edge devices could cooperate and help each other to provide enough computation resources. For example, if a camera needs to perform image recognition task, it could partition the CNN model by layers, and transmit the partitioned tasks to other devices nearby. In this scenario, a cluster of edge devices could be organised as a virtual edge server, which could execute inference tasks from both inside and outside of the cluster. Hadidi *et al.* propose Musical Chair, an offloading framework that harvests available computing resources in an IoT network for cooperation [373]. In Musical Chair, the authors develop

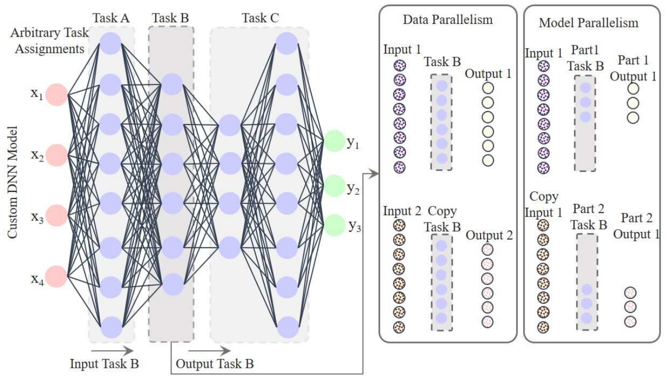


Fig. 37. The parallelism structure of Musical Chair. Task B is a layer-level task, which are further partitioned into two sub-tasks on two devices. These two devices adopt different input to double the system throughput.

data parallelism and model parallelism scheme to speed up the inference. Data parallelism refers to duplicating devices that performs the same task whilst model parallelism is about performing different sub-tasks of a task on different devices. Fig. 37 shows the parallel structure for a layer-level task. Talagala *et al.* use a graph-based overlay network to specify the pipeline dependencies in neural networks and propose a server/agent architecture to schedule computing tasks amongst edge devices in similar scenarios [374].

Coninck *et al.* develop DIANNE, a modular distributed framework, which treats neural network layers as directed graphs [375]. As shown in Fig. 38, each module provides forward and backward methods, corresponding to feedforward and back-propagation, respectively. Each module is a unit deployed on an edge device. There is a job scheduler component, which assigns learning jobs to devices with spare resources. Fukushima *et al.* propose the MicroDeep framework, which assigns neurons of CNN to wireless sensors for image recognition model training [376]. The structure of MicroDeep is similar to DIANNE. Each CNN unit is allocated to a wireless sensor, which executes the training of the unit parameters. In feedforward phase, sensors exchange their output data. Once a sensor receives the necessary input, it executes its unit and broadcasts its output for subsequent layer. If a sensor with an output layer unit obtains its input and ground-truth, it starts the back-propagation phase. They adopt a 2D coordinate based approach to approximately allocate a CNN unit to sensors.

Distributed solo learning enables edge devices or edge servers to train models with local data. Consequently, each

TABLE XII
LITERATURE SUMMARY OF D2D OFFLOADING STRATEGY.

Ref.	Enabler	Problem	Object	Baseline	Performance
[370]	Static partition	Model partition	Speedup, save energy	Wearable-only strategy	23× faster 85.5% less energy
[372]	Incremental training	Model Adaptability	Improve accuracy	Cloud-based scheme	3.3× faster, 30%-70% energy-saving
[373]	Data, task parallelism	Offloading among peers	Computing power	Local inference	90× faster, 200× less energy
[375]	Modularization	Offloading among peers	Modular architecture	Local inference	4.5× faster
[376]	Disassembling CNN	Neuron assignment	Training on sensor	Standard CNN	95.57% accuracy
[377]	Bayesian	Knowledge retrieval	Routing strategy	N/A	95% accuracy

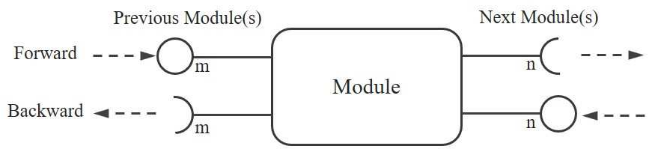


Fig. 38. The illustration of a DIANNE module. Each module has references to its predecessors and successors for feedforward and back-propagation during training.

model may become local experts that are good at predicting local phenomena. For example, RSUs use local trained models to predict local traffic condition. However, users are interested in the traffic condition of places they plan to visit, in addition to the local traffic condition. Bach *et al.* propose a routing strategy to forward the queries to devices that have the specific knowledge [377]. The strategy is similar to the routing strategy in TCP/IP networks. Each device maintains a routing table to guide the forwarding. The strategy achieve 95% accuracy in their experiments. However, latency is a big problem in such frameworks.

D. Hybrid offloading

The hybrid offloading architecture, also named osmotic computing [539], refers to the computing paradigm that is supported by the seamless collaboration between edge and cloud computing resources, along with the assistance of data transfer protocols. The hybrid computing architecture takes advantage of cloud, edge, and mobile devices in a holistic manner. There are some efforts focusing on distributing deep learning models in such environments.

Morshed *et al.* investigate 'deep osmosis' and analyses the challenges involved with the holistic distributed deep learning architecture, as well as the data and resource architecture [378]. Teerapittayanon *et al.* propose distributed deep neural networks (DDNNs) based on the holistic computing architecture, which maps sections of a DNN onto a distributed computing hierarchy [379]. All sections are jointly trained in the cloud to minimise communication and resource usage for edge devices. During inference, each edge device performs local computation and then all outputs are aggregated to output the final results.

There is always the risk that the physical nodes i.e., edge devices and edge servers may fail, which results in the failure of DNN units deployed on these physical nodes. Yousefpour *et al.* introduce 'deepFogGuard' to make the distributed DNN inference failure-resilient [380]. Similar to residual connections [268], which skips DNN layers to reduce the runtime, 'deepFogGuard' skips physical nodes to minimise the impact of failed DNN units. The authors also verify the resilience of 'deepFogGuard' on sensing and vision applications.

E. Applications

There exists some works applying the above mentioned offloading strategy to practical applications, such as face recognition [381], intelligent transportation [382], smart industry [383], smart city [384], and healthcare [385]–[387]. Specifically, Muslim *et al.* design a D2E offloading based face recognition system [381]. The authors divide the face recognition process in to two phases: face detection and face recognition. The first phase is executed on smartphone, while the second phase is executed either locally or on edge server. They develop a reinforcement learning based algorithm to make the offloading decision based on different environments. Ferdowsi *et al.* design an edge-centric architecture for intelligent transportation, where roadside smart sensors and vehicles could work as edge servers to provide low latency deep learning based services [382]. Li *et al.* proposes a deep learning based classification model to detect defective products in assembly lines, which leverages an edge server to provide computing resources [383]. Tang *et al.* develop a hierarchical distributed framework to support data intensive analytics in smart cities [384]. They develop a pipeline monitoring system for anomaly detection. Edge devices and servers provide computing resources for the execution of these detection models.

People care more about heathy eating habits currently. Nutrition monitoring systems provide scientific suggestions on eating to help people stay in health via monitoring the food they eat everyday. Liu *et al.* design an edge based food recognition system for dietary assessment, which splits the recognition tasks between nearby edge devices and cloud server to solve the latency and energy consumption problem [385]. Javadi *et al.* further develop nutritional ingredient analysis system based on food recognition to provide better nutrition monitoring service for users [386]. The structure is similar to [385]. Muhammed *et al.* develop a ubiquitous healthcare framework, called UbeHealth, which makes full use of deep learning, big data, and computing resources [387]. They use big data to predict the network traffic, which in turn is used to assist the edge server to make the optimal offloading decision.

VIII. FUTURE DIRECTIONS AND OPEN CHALLENGES

We present a thorough and comprehensive survey on the literature surrounding edge intelligence. The benefits of edge intelligence are obvious - it paves the way for the last mile of AI and to provide high-efficient intelligent services for people, which significantly lessens the dependency on central cloud servers, and can effectively protect data privacy. It is worth recapping that there are still some unsolved open challenges in realising edge intelligence. It is crucial to identify and analyze these challenges and seek for novel theoretical and technical solutions. In this view, we discuss some prominent challenges in edge intelligence along some possible solutions. These challenges include data scarcity at edge, data consistency on edge devices, bad adaptability of statically trained model, privacy and security issues, incentive mechanism, and the relationship with future 6G network.

A. *Data scarcity at edge*

For most machine learning algorithms, especially supervised machine learning, the high performance depends on sufficiently high-quality training instances. However, it often does not work in edge intelligence scenarios, where the collected data is sparse and unlabeled, e.g., in HAR and speech recognition applications. Different from traditional cloud based intelligent services, where the training instances are all gathered in a central database, edge devices use the self-generated data or the data captured from surrounding environments to generate models. High-quality training instances, e.g., good image features are lacking in such datasets. Most existing works ignore this challenge, assuming that the training instances are of good quality. Moreover, the training dataset is often unlabeled. Some works [145], [150] propose to use active learning to solve the problem of unlabeled training instances, which requires manual intervention for annotation. Such an approach could only be used in scenarios with few instances and classifications. Federated learning approaches leverage the decentralised characteristic of data to effectively solve the problem. However, federated learning is only suitable for collaboration training, instead of the solo training needed for personalised models.

We discuss several possible solutions for this problem as follows.

- Adopt shallow models, which could be trained with only a small dataset. Generally, the simpler the machine learning algorithm is, the better the algorithm will learn from the small datasets. A simple model, e.g., Naive Bayes, linear model, and decision tree, are enough to deal with the problem in some scenarios, compared with complicated models, e.g., neural network, since they are essentially trying to learn less. Hence, adopting an appropriate model should be taken into consideration when dealing with practical problems.
- Incremental learning based methods. Edge devices could re-train a commonly-used pre-trained model in an incremental fashion to accommodate their new data. In such a manner, only few training instances are required to generate a customised model.
- Transfer learning based methods, e.g., few-shot learning. Transfer learning uses the learned knowledge from other models to enhance the performance of a related model, typically avoiding the cold-start problem and reducing the amount of required training data. Hence, transfer learning could be a possible solution, when there is not enough target training data, and the source and target domains have some similarities.
- Data augmentation based methods. Data augmentation enables a model to be more robust by enriching data during the training phase [540]. For example, increasing the number of images without changing the semantic meaning of the labels through flipping, rotation, scaling, translation, cropping, etc. Through the training on augmented data, the network would be invariant to these deformations and have better performance to unseen data.

B. *Data consistency on edge devices*

Edge intelligence based applications, e.g., speech recognition, activity recognition, emotion recognition, etc., usually collect data from large amounts of sensors distributed at the edge network. Nevertheless, the collected data may not be consistent. Two factors contribute to this problem: different sensing environments, and sensor heterogeneity. The environment (e.g., street and library) and its conditions (e.g., raining, windy) add background noise to the collected sensor data, which may have an impact on the model accuracy. The heterogeneity of sensors (e.g., hardware and software) may also cause the unexpected variation in their collected data. For example, different sensors have different sensitivities, sampling rates, and sensing efficiencies. Even the sensor data collected from the same source may vary on different sensors. Consequently, the variation of the data would result in the variation on the model training, e.g., the parameters of features [502], [541], [542]. Such variation is still a challenge for existing sensing applications.

This problem could be solved easily if the model is trained in a centralised manner. The centralised large training set guarantees that the invariant features to the variations could be learned. However, this is not the scope of edge intelligence. Future efforts of this problem should focus on how to block the negative effect of the variation on model accuracy. To this end, two possible research directions maybe considered: data augmentation, and representation learning. Data augmentation could enrich the data during the model training process to enable the model to be more robust to noise. For example, adding various kinds of background noises to block the variation caused by the environments in speech recognition applications on mobile devices. Similarly, the noise caused by the hardware of sensors could also be added to deal with the inconsistency problem. Through the training of the augmented data, the models are more robust with these variations.

Data representation heavily affects the performance of models. Representation leaning focuses on learning the representation of data to extract more effective features when building models [431], which could also be used to hide the differences between different hardware. For this problem, if we could make a ‘translation’ on the representations between two sensors which are working on the same data source, the performance of the model would be improved significantly. Hence, representation learning is a promising solution to diminish the impact of data inconsistency. Future efforts could be made on this direction, e.g., design more effective processing pipelines and data transformations.

C. Bad adaptability of statically trained model

In most edge intelligence based AI applications, the model is first trained on a central server, then deployed on edge devices. The trained model will not be retrained, once the training procedure is finished. These statically trained models cannot effectively deal with the unknown new data and tasks in unfamiliar environments, which results in low performance and bad user experience. On the other hands, for models trained with a decentralised learning manner, only the local experience is used. Consequently, such models may become experts only in their small local areas. When the serving area broadens, the quality of service decreases.

To cope with this problem, two possible solutions may be considered: lifelong machine learning and knowledge sharing. Lifelong machine learning (LML) [543] is an advanced learning paradigm, which enables continuous knowledge accumulation and self-learning on new tasks. Machines are taught to learn new knowledge by themselves based on previously learned knowledge, instead of being trained by humans. LML is slightly different from meta learning [544], which enables machines to automatically learn new models. Edge devices with a series of learned tasks could use LML to adapt to changing environments and to deal with unknown data. It is worth recapping that the LML is not primarily designed for edge devices, which means that the machines are expected to be computationally powerful. Accordingly, model design, model compression, and offloading strategies should be also considered if LML is applied.

Knowledge sharing [545] enables the knowledge communication between different edge servers. When there is a task submitted to an edge server that does not have enough knowledge to provide a good service, the server could send knowledge queries to other edge servers. Since the knowledge is allocated on different edge servers, the server with the required knowledge responds to the query and performs the task for users. A knowledge assessment method and knowledge query system are required in such a knowledge sharing paradigm.

D. Privacy and security issues

To realise edge intelligence, heterogeneous edge devices and edge servers are required to work collaboratively to provide computing powers. In this procedure, the locally cached data and computing tasks (either training or inference tasks) might be sent to unfamiliar devices for further processing. The data may contain users' private information, e.g. photos and tokens, which leads to the risk of privacy leakage and attacks from malicious users. If the data is not encrypted, malicious users could easily obtain private information from the data. Some efforts [347], [349], [350], [530] propose to do some preliminary processing locally, which could hide private information and reduce the amount of transmitted data. However, it is still possible to extract private information from the processed data [185]. Moreover, malicious users could also attack and control a device that provides computing power through inserting a virus in the computing tasks. The key challenge is the lack of relevant privacy preserving and security protocols or mechanisms to protect users' privacy and security from being attacked.

Credit system maybe a possible solution. This is similar with the credit system used in banks, which authenticates each user participated in the system and checks their credit information. Users with bad credit records would be deleted from the system. Consequently, all devices that provide computing powers are credible and all users are safe.

Encryption could be used to protect privacy, which is already applied in some works [154], [155], [546]–[548]. However, the encrypted data need to be decrypted before the training or inference tasks are executed, which requires an increase in the amount of computation needed. Homomorphic encryption may be a possible solution to solve the problem [188]. Homomorphic encryption refers to an encryption method that allows direct computation on ciphertexts and generate encrypted results. After decryption, the result is the same as the result achieved by computation on the unencrypted data. Hence, by applying homomorphic encryption, the training or inference task could be directly executed on encrypted data. However, the encryption on edge device also requires intensive computation. Some works [549], [550] have investigated light-weight homomorphic encryption algorithms to reduce computation overhead. Future work may focus on the balance between encryption overhead and the security efficiency in edge intelligence.

E. Incentive mechanism

Data collection and model training/inference are two utmost important steps for edge intelligence. For data collection, it is challenging to ensure the quality and usability of information of the collected data. Data collectors consume their own resources, e.g., battery, bandwidth, and the time to sense and collect data. It is not realistic to assume that all data collectors are willing to contribute, let alone for preprocessing data cleaning, feature extraction and encryption, which further consumes more resources. For model training/inference in a collaborative manner, all participants are required to unselfishly work together for a given task. For example, the architecture proposed in [143] consists of one master and multi workers. Workers recognise objects in a particular mobile visual domain and provides training instances for masters through pipelines. Such architecture works in private scenarios, e.g., at home, where all devices are inherently motivated to collaboratively create a better intelligent model for their master, i.e., their owner. However, it would not work well in public scenarios, where the master initialises a task and allocates sub-tasks to unfamiliar participants. In this context, additional incentive issue arises, which is not typically considered in smart environments where all devices are not under the ownership of a single master. Participants need to be incentivised to perform data collection and task execution.

Reasonable incentive mechanisms should be considered for future efforts. On one hand, participants have different missions, e.g., data collection, data processing, and data analysis, which have different resource consumptions. All participants hope to get as much reward as possible. On the other hand, the operator hopes to achieve the best model accuracy with as a low cost as possible. The challenges of designing the optimal incentive mechanism are how to quantify the workloads of different missions to match corresponding rewards and how to jointly optimise these two conflicting objectives. Future efforts could focus on addressing these challenges.

F. Edge intelligence with 6G

The fifth-generation (5G) wireless network is being applied all over the world. The complete 5G network can provide high peak data transmission rate, ultra-low communication latency, and ubiquitous connections. However, with the fast increasing demand on communication, it is forecasted that the capacity of 5G network will be surpassed in 2030 [551], [552].

6G network, as the next evolution of 5G network, has been proposed to meet the requirements that are out of the capability of 5G. The commonly visioned aspects of 6G include global coverage, more spectrum resource, intelligent communication, high-level network security [553]–[555]. Global coverage means that users can access the Internet anytime and anywhere in the world. Current 5G networks are mainly deployed in metropolises, the coverage is not good in remote areas. In 6G networks, satellite networks with high-bandwidth and high-quality communication channels will be established to solve the coverage problem. Spectrum resource is scarce. Terahertz and optical frequency bands may be considered in 6G networks to broaden spectrum resource.

In scenarios network resources are fixed, optimization of resource allocation will improve the quality of service and users' experience. However, the optimization problems are usually NP-hard problem with multiple objectives under complex constraints, which are hard to be solved. Moreover, it is usually difficult to formulate these problems with accurate mathematical models. AI algorithms, especially deep learning algorithms, are of high adaptivity, which could automatically extract features from data and learn the optimal resource allocation scheme. In addition, network resource could be also optimized to improve efficiency and avoid congestion through learning the traffic pattern.

However, AI models are data-driven and usually require massive data to train models. Traditionally cloud-based AI requires to upload giant amount of data from edge to cloud, which may decrease the benefit of broadened bandwidth. Moreover, there may be large amount of private information involved in the uploaded data, which is also contrary to vision of enhanced security of 6G network. A possible solution is to bring edge intelligence to each node, e.g., infrastructure in the network, as well as the clusters of nodes. These nodes could learn with local data and possibly share the knowledge with others to collaboratively learn an optimal model with good performance.

IX. Conclusions

In this paper, we present a thorough and comprehensive survey on the literature surrounding edge intelligence. Specifically, we identify critical components of edge intelligence: edge caching, edge training, edge inference, and edge offloading. Based on this, we provide a systematic classification of literature by reviewing research achievements for each components and present a systematical taxonomy according to practical problems, adopted techniques, application goals, etc. We compare, discuss and analyse literature in the taxonomy from multi-dimensions, i.e., adopted techniques, objectives, performance, advantages and drawbacks, etc. Moreover, we also discuss important open issues and present possible theoretical research directions. Concerning the era of edge intelligence, We believe that this is only the tip of iceberg. Along with the explosive development trend of IoT and AI, we expect that more and more research efforts would be carried out to completely realize edge intelligence in the following decades.

X. Acknowledgements

This work was supported in part by National Key R&D Program of China under Grant 2020YFA0711400, 2020AAA0106000, the National Natural Science Foundation of China under U1936217, 61971267, 61972223, 61941117, 61861136003, Beijing Natural Science Foundation under L182038, Beijing National Research Center for Information Science and Technology under 20031887521, and Academy of Finland under projects 319669, 319670, 325570, 326305, 325774, 335934. We would also thank Mr. Jacky Cao for his diligent proofreading of this article.

References

[1] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," 2015.

[2] Y. Sun, D. Liang, X. Wang, and X. Tang, "Deepid3: Face recognition with very deep neural networks," *arXiv preprint arXiv:1502.00873*, 2015.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[4] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[5] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" in *Advances in neural information processing systems*, 2017, pp. 5574–5584.

[6] H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, "Augmented reality meets deep learning for car instance segmentation in urban scenes," in *British machine vision conference*, vol. 1, 2017, p. 2.

[7] W. Huang, G. Song, H. Hong, and K. Xie, "Deep architecture for traffic flow prediction: deep belief networks with multitask learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 2191–2201, 2014.

[8] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with big data: a deep learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865–873, 2014.

[9] P. Zhou, T. Braud, A. Alhilal, P. Hui, and J. Kangasharju, "Erl: Edge based reinforcement learning for optimized urban traffic light control," in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019, pp. 849–854.

[10] Z. Fang, F. Fei, Y. Fang, C. Lee, N. Xiong, L. Shu, and S. Chen, "Abnormal event detection in crowded scenes based on deep learning," *Multimedia Tools and Applications*, vol. 75, no. 22, pp. 14 617–14 639, 2016.

[11] C. Potes, S. Parvaneh, A. Rahman, and B. Conroy, "Ensemble of feature-based and deep learning-based classifiers for detection of abnormal heart sounds," in *2016 Computing in Cardiology Conference (CinC)*. IEEE, 2016, pp. 621–624.

[12] "Cisco visual networking index: Global mobile data traffic forecast update (2017–2022)," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html, 2016.

[13] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[14] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[15] 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, Oct 2016.

[16] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[17] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[18] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

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

[20] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, 2019.

[21] C. Li, Y. Xue, J. Wang, W. Zhang, and T. Li, "Edge-oriented computing paradigms: A survey on architecture design and system management," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–34, 2018.

[22] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[23] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*. IEEE, 2016, pp. 20–26.

[24] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A survey on edge computing systems and tools," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1537–1562, 2019.

[25] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[26] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[27] Z. Wang, Y. Cui, and Z. Lai, "A first look at mobile intelligence: Architecture, experimentation and challenges," *IEEE Network*, 2019.

[28] H. Khelifi, S. Luo, B. Nour, A. Sellami, H. Moungla, S. H. Ahmed, and M. Guizani, "Bringing deep learning at the edge of information-centric internet of things," *IEEE Communications Letters*, vol. 23, no. 1, pp. 52–55, 2018.

[29] N. D. Lane and P. Warden, "The deep (learning) transformation of mobile and embedded computing," *Computer*, vol. 51, no. 5, pp. 12–16, 2018.

[30] F. Chen, Z. Dong, Z. Li, and X. He, "Federated meta-learning for recommendation," *arXiv preprint arXiv:1802.07876*, 2018.

[31] Y. Chen, J. Wang, C. Yu, W. Gao, and X. Qin, "Fedhealth: A federated transfer learning framework for wearable healthcare," *arXiv preprint arXiv:1907.09173*, 2019.

[32] E. Peltonen, M. Bennis, M. Capobianco, M. Debbah, A. Ding, F. Gil-Castiñeira, M. Jurmu, T. Karvonen, M. Kelanti, A. Kliks *et al.*, "6g white paper on edge intelligence," *arXiv preprint arXiv:2004.14850*, 2020.

[33] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: Wireless communication meets machine learning," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 19–25, 2020.

[34] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. IEEE, 2015, pp. 73–78.

[35] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, 2014, pp. 68–81.

[36] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 82–88, 2017.

[37] V. Radu, C. Tong, S. Bhattacharya, N. D. Lane, C. Mascolo, M. K. Marina, and F. Kawsar, "Multimodal deep learning for activity and context recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, p. 157, 2018.

[38] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proceedings of the 2015 international workshop on internet of things towards applications*. ACM, 2015, pp. 7–12.

[39] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," *Google Research Blog*, vol. 3, 2017.

[40] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, p. 12, 2019.

[41] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief, "Communication-efficient edge ai: Algorithms and systems," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2167–2191, 2020.

[42] J. Wang, B. Cao, P. Yu, L. Sun, W. Bao, and X. Zhu, "Deep learning towards mobile applications," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1385–1393.

[43] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2020.

[44] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[45] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[46] M. Xu, X. Liu, Y. Liu, and F. X. Lin, "Accelerating convolutional neural networks for continuous mobile vision via cache reuse," *arXiv preprint arXiv:1712.01670*, 2017.

[47] P. Guo, B. Hu, R. Li, and W. Hu, "Foggycache: Cross-device approximate computation reuse," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 19–34.

[48] Google Street View Image API, *https://developers.google.com/maps/documentation/streetview/intro*, 2019.

[49] R. Huitl, G. Schroth, S. Hilsenbeck, F. Schweiger, and E. Steinbach, "Tumindoor: An extensive image and point cloud dataset for visual indoor localization and mapping," in *2012 19th IEEE International Conference on Image Processing*. IEEE, 2012, pp. 1773–1776.

[50] L. Kong, Z. Wu, G. Chen, M. Qiu, S. Mumtaz, and J. J. Rodrigues, "Crowdsensing-based cross-operator switch in rail transit systems," *IEEE Transactions on Communications*, vol. 68, no. 12, pp. 7938–7947, 2020.

[51] Z. Wu, L. Kong, G. Chen, M. K. Khan, S. Mumtaz, and J. J. Rodrigues, "Crowdswitch: Crowdsensing based switch between multiple cellular operators in subways," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[52] M. Elhamshary, M. Youssef, A. Uchiyama, H. Yamaguchi, and T. Higashino, "Transitlabel: A crowd-sensing system for automatic labeling of transit stations semantics," in *Proceedings of the 14th annual international conference on mobile systems, applications, and services*, 2016, pp. 193–206.

[53] J. Wang, J. Su, and R. Hua, "Design of a smart independent smoke sense system based on nb-iot technology," in *2019 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*. IEEE, 2019, pp. 397–400.

[54] H. Cao, H. Yao, H. Cheng, and S. Lian, "A solution for data collection of large-scale outdoor internet of things based on uav and dynamic clustering," in *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 9. IEEE, 2020, pp. 2133–2136.

[55] B. Jiang, G. Huang, T. Wang, J. Gui, and X. Zhu, "Trust based energy efficient data collection with unmanned aerial vehicle in edge network," *Transactions on Emerging Telecommunications Technologies*, p. e3942, 2020.

[56] M. Shen, A. Liu, G. Huang, N. N. Xiong, and H. Lu, "Attdc: An active and trace-able trust data collection scheme for industrial security in smart cities," *IEEE Internet of Things Journal*, 2021.

[57] X. Li, J. Tan, A. Liu, P. Vijayakumar, N. Kumar, and M. Alazab, "A novel uav-enabled data collection scheme for intelligent transportation system through uav speed control," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[58] X. Li and D. W. Goldberg, "Toward a mobile crowdsensing system for road surface assessment," *Computers, Environment and Urban Systems*, vol. 69, pp. 51–62, 2018.

[59] F. J. Villanueva, D. Villa, M. J. Santofimia, J. Barba, and J. C. Lopez, "Crowdsensing smart city parking monitoring," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 751–756.

[60] C. Wang, Z. Xie, L. Shao, Z. Zhang, and M. Zhou, "Estimating travel speed of a road section through sparse crowdsensing data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 9, pp. 3486–3495, 2018.

[61] A. A. Nauman Mazhar, A. Razzaq, K. Abbas, M. Adnan, H. Abdullah, and N. Rasheed, "Mobile crowdsensing application of road condition detection," *Urdu News Headline, Text Classification by Using Different Machine Learning Algorithms*, p. 25, 2019.

[62] F. Kalim, J. P. Jeong, and M. U. Ilyas, "Crater: A crowd sensing application to estimate road conditions," *Ieee access*, vol. 4, pp. 8317–8326, 2016.

[63] V. Singh, D. Chander, U. Chhaparia, and B. Raman, "Safestreet: An automated road anomaly detection and early-warning system using mobile crowdsensing," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*. IEEE, 2018, pp. 549–552.

[64] C. Leonardi, A. Cappellotto, M. Caraviello, B. Lepri, and F. Antonelli, "Secondnose: an air quality mobile crowdsensing system," in *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*, 2014, pp. 1051–1054.

[65] L. Liu, W. Liu, Y. Zheng, H. Ma, and C. Zhang, "Third-eye: A mobilephone-enabled crowdsensing system for air quality monitoring," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, pp. 1–26, 2018.

[66] M. Marjanović, S. Grubeša, and I. P. Žarko, "Air and noise pollution monitoring in the city of zagreb by using mobile crowdsensing," in *2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2017, pp. 1–5.

[67] F. Seraj, B. J. Van Der Zwaag, A. Dilo, T. Luarasi, and P. Havinga, "Roads: A road pavement monitoring system for anomaly detection using smart phones," in *Big data analytics in the social and ubiquitous context*. Springer, 2015, pp. 128–146.

[68] H. Bello-Salau, A. Aibinu, A. Onumanyi, E. Onwuka, J. Dukiya, and H. Ohize, "New road anomaly detection and characterization algorithm for autonomous vehicles," *Applied Computing and Informatics*, 2020.

[69] X. Li, D. Huo, D. W. Goldberg, T. Chu, Z. Yin, and T. Hammond, "Embracing crowdsensing: An enhanced mobile sensing solution for road anomaly detection," *ISPRS International Journal of Geo-Information*, vol. 8, no. 9, p. 412, 2019.

[70] A. Mehrabi and K. Kim, "General framework for network throughput maximization in sink-based energy harvesting wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 7, pp. 1881–1896, 2016.

[71] H. Salarian, K.-W. Chin, and F. Naghdy, "An energy-efficient mobile-sink path selection strategy for wireless sensor networks," *IEEE Transactions on vehicular technology*, vol. 63, no. 5, pp. 2407–2419, 2013.

[72] Y. Liu, K.-Y. Lam, S. Han, and Q. Chen, "Mobile data gathering and energy harvesting in rechargeable wireless sensor networks," *Information Sciences*, vol. 482, pp. 189–209, 2019.

[73] M. Bonola, L. Bracciale, P. Loreti, R. Amici, A. Rabuffi, and G. Bianchi, "Opportunistic communication in smart city: Experimental insight with small-scale taxi fleets as data carriers," *Ad Hoc Networks*, vol. 43, pp. 43–55, 2016.

[74] Y. Luo, X. Zhu, and J. Long, "Data collection through mobile vehicles in edge network of smart city," *IEEE Access*, vol. 7, pp. 168 467–168 483, 2019.

[75] B. Brik, N. Lagraa, A. Lakas, H. Cherroun, and A. Cheddad, "Ecdgp: extended cluster-based data gathering protocol for vehicular networks," *Wireless Communications and Mobile Computing*, vol. 16, no. 10, pp. 1238–1255, 2016.

[76] F. van Wyk, Y. Wang, A. Khojandi, and N. Masoud, "Real-time sensor anomaly detection and identification in automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1264–1276, 2019.

[77] B. V. Philip, T. Alpcan, J. Jin, and M. Palaniswami, "Distributed real-time iot for autonomous vehicles," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1131–1140, 2018.

[78] Z. He, J. Cao, and X. Liu, "High quality participant recruitment in vehicle-based crowdsourcing using predictable mobility," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 2542–2550.

[79] Y. Liu, J. Niu, and X. Liu, "Comprehensive tempo-spatial data collection in crowd sensing using a heterogeneous sensing vehicle selection method," *Personal and Ubiquitous Computing*, vol. 20, no. 3, pp. 397–411, 2016.

[80] X. Zhu, Y. Luo, A. Liu, W. Tang, and M. Z. A. Bhuiyan, "A deep learning-based mobile crowdsensing scheme by predicting vehicle mobility," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[81] Z. Lin, Y. Lai, X. Gao, G. Li, T. Wang, and G. Huang, "Data gathering in urban vehicular network based on daily movement patterns," in *2016 11th International Conference on Computer Science & Education (ICCSE)*. IEEE, 2016, pp. 641–646.

[82] Y. Ren, T. Wang, S. Zhang, and J. Zhang, "An intelligent big data collection technology based on micro mobile data centers for crowdsensing vehicular sensor network," *Personal and Ubiquitous Computing*, pp. 1–17, 2020.

[83] S. Huang, A. Liu, S. Zhang, T. Wang, and N. Xiong, "Bd-vte: a novel baseline data based verifiable trust evaluation scheme for smart network systems," *IEEE Transactions on Network Science and Engineering*, 2020.

[84] N. Javaid, S. Cheema, M. Akbar, N. Alrajeh, M. S. Alabed, and N. Guizani, "Balanced energy consumption based adaptive routing for iot enabling underwater wsns," *IEEE Access*, vol. 5, pp. 10 040–10 051, 2017.

[85] M. Chen, T. Wang, K. Ota, M. Dong, M. Zhao, and A. Liu, "Intelligent resource allocation management for vehicles network: An a3c learning approach," *Computer Communications*, vol. 151, pp. 485–494, 2020.

[86] O. Yan, A. Liu, N. Xiong, and T. Wang, "An effective early message ahead join adaptive data aggregation scheme for sustainable iot," *IEEE Transactions on Network Science and Engineering*, 2020.

[87] Q. Wang, H.-N. Dai, Q. Wang, M. K. Shukla, W. Zhang, and C. G. Soares, "On connectivity of uav-assisted data acquisition for underwater internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5371–5385, 2020.

[88] N. Ilyas, T. A. Alghamdi, M. N. Farooq, B. Mehboob, A. H. Sadiq, U. Qasim, Z. A. Khan, and N. Javaid, "Aedg: Auv-aided efficient data gathering routing protocol for underwater wireless sensor networks," *Procedia Computer Science*, vol. 52, pp. 568–575, 2015.

[89] X. Zhuo, M. Liu, Y. Wei, G. Yu, F. Qu, and R. Sun, "Auv-aided energy-efficient data collection in underwater acoustic sensor networks," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 010–10 022, 2020.

[90] N. Ilyas, M. Akbar, R. Ullah, M. Khalid, A. Arif, A. Hafeez, U. Qasim, Z. A. Khan, and N. Javaid, "Sedg: Scalable and efficient data gathering routing protocol for underwater wsns," *Procedia Computer Science*, vol. 52, pp. 584–591, 2015.

[91] S. Cai, Y. Zhu, T. Wang, G. Xu, A. Liu, and X. Liu, "Data collection in underwater sensor networks based on mobile edge computing," *IEEE Access*, vol. 7, pp. 65 357–65 367, 2019.

[92] I. Psaras, O. Ascigil, S. Rene, G. Pavlou, A. Afanasyev, and L. Zhang, "Mobile data repositories at the edge," in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[93] A.-C. Nicolaescu, O. Ascigil, and I. Psaras, "Edge data repositories-the design of a store-process-send system at the edge," in *Proceedings of the 1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms*, 2019, pp. 41–47.

[94] J. Liu, M. L. Curry, C. Maltzahn, and P. Kufeldt, "Scale-out edge storage systems with embedded storage nodes to get better availability and cost-efficiency at the same time," in *3rd* {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 20)*, 2020.

[95] A. C. Baktir, A. Ozgovde, and C. Ersoy, "Enabling service-centric networks for cloudlets using sdn," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 344–352.

[96] A.-C. Nicolaescu, S. Mastorakis, and I. Psara, "Store edge networked data (send): A data and performance driven edge storage framework," in *Proc. IEEE Conf. Comput. Commun.(INFOCOM)*, 2021, pp. 1–11.

[97] J. Xie, C. Qian, D. Guo, X. Li, S. Shi, and H. Chen, "Efficient data placement and retrieval services in edge computing," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1029–1039.

[98] C. Li, J. Bai, and J. Tang, "Joint optimization of data placement and scheduling for improving user experience in edge computing," *Journal of Parallel and Distributed Computing*, vol. 125, pp. 93–105, 2019.

[99] X. Du, S. Tang, Z. Lu, J. Wet, K. Gai, and P. C. Hung, "A novel data placement strategy for data-sharing scientific workflows in heterogeneous edge-cloud computing environments," in *2020 IEEE International Conference on Web Services (ICWS)*. IEEE, 2020, pp. 498–507.

[100] L. Tseng, T. Higuchi, and O. Altintas, "When cars meet distributed computing: Data storage as an example," *arXiv preprint arXiv:1711.02014*, 2017.

[101] F. Dressler, P. Handle, and C. Sommer, "Towards a vehicular cloud-using parked vehicles as a temporary network and storage infrastructure," in *Proceedings of the 2014 ACM international workshop on Wireless and mobile technologies for smart cities*, 2014, pp. 11–18.

[102] E. Lee, E.-K. Lee, M. Gerla, and S. Y. Oh, "Vehicular cloud networking: architecture and design principles," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 148–155, 2014.

[103] M. Gerla, "Vehicular cloud computing," in *2012 The 11th annual mediterranean ad hoc networking workshop (Med-Hoc-Net)*. IEEE, 2012, pp. 152–155.

[104] T. Higuchi, S. Ucar, C.-H. Wang, and O. Altintas, "Vehicular micro cloud as an enabler of intelligent intersection management," in *2020 IEEE Vehicular Networking Conference (VNC)*. IEEE, 2020, pp. 1–2.

[105] T. Higuchi, J. Joy, F. Dressler, M. Gerla, and O. Altintas, "On the feasibility of vehicular micro clouds," in *2017 IEEE Vehicular Networking Conference (VNC)*. IEEE, 2017, pp. 179–182.

[106] F. Hagenauer, C. Sommer, T. Higuchi, O. Altintas, and F. Dressler, "Vehicular micro cloud in action: On gateway selection and gateway handovers," *Ad Hoc Networks*, vol. 78, pp. 73–83, 2018.

[107] ——, "Poster: Using clusters of parked cars as virtual vehicular network infrastructure," in *2016 IEEE Vehicular Networking Conference (VNC)*. IEEE, 2016, pp. 1–2.

[108] F. Dressler, G. S. Pannu, F. Hagenauer, M. Gerla, T. Higuchi, and O. Altintas, "Virtual edge computing using vehicular micro clouds," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 537–541.

[109] C. X. Mavromoustakis and G. Mastorakis, "Analysis of vehicular storage and dissemination services based on floating content," in *Mobile Networks and Management: 6th International Conference, MONAMI 2014, Würzburg, Germany, September 22-26, 2014, Revised Selected Papers*, vol. 141. Springer, 2015, p. 387.

[110] T. Higuchi, F. Dressler, and O. Altintas, "How to keep a vehicular micro cloud intact," in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*. IEEE, 2018, pp. 1–5.

[111] G. S. Pannu, F. Hagenauer, T. Higuchi, O. Altintas, and F. Dressler, "Keeping data alive: Communication across vehicular micro clouds," in *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2019, pp. 1–9.

[112] S. Wolfert, L. Ge, C. Verdouw, and M.-J. Bogaardt, "Big data in smart farming–a review," *Agricultural systems*, vol. 153, pp. 69–80, 2017.

[113] K. Bhargava, S. Ivanov, W. Donnelly, and C. Kulatunga, "Using edge analytics to improve data collection in precision dairy farming," in *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*. IEEE, 2016, pp. 137–144.

[114] K. A. Khaliq, O. Chughtai, A. Shahwani, A. Qayyum, and J. Pannek, "Road accidents detection, data collection and data analysis using v2x communication and edge/cloud computing," *Electronics*, vol. 8, no. 8, p. 896, 2019.

[115] B. Nour, S. Mastorakis, and A. Mtibaa, "Compute-less networking: Perspectives, challenges, and opportunities," *IEEE Network*, vol. 34, no. 6, pp. 259–265, 2020.

[116] J. Lee, A. Mtibaa, and S. Mastorakis, "A case for compute reuse in future edge systems: An empirical study," in *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019, pp. 1–6.

[117] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "Deepcache: Principled cache for mobile deep vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 129–144.

[118] N. L. HUYNH, R. K. Balan, and Y. Lee, "Deepmon-building mobile gpu deep learning models for continuous vision applications," 2017.

[119] L. Cavigelli and L. Benini, "Cbinfer: Exploiting frame-to-frame locality for faster convolutional network inference on video streams," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 5, pp. 1451–1465, 2019.

[120] L. Cavigelli, P. Degen, and L. Benini, "Cbinfer: Change-based inference for convolutional neural networks on video data," in *Proceedings of the 11th International Conference on Distributed Smart Cameras*, 2017, pp. 1–8.

[121] P. O'Connor and M. Welling, "Sigma delta quantized networks," *arXiv preprint arXiv:1611.02024*, 2016.

[122] Y.-W. Huang, C.-Y. Chen, C.-H. Tsai, C.-F. Shen, and L.-G. Chen, "Survey on block matching motion estimation algorithms and architectures with new results," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 42, no. 3, pp. 297–320, 2006.

[123] M. Buckler, P. Bedoukian, S. Jayasuriya, and A. Sampson, "Eva^2: Exploiting temporal redundancy in live computer vision," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 533–546.

[124] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2015, pp. 155–168.

[125] S. Naderiparizi, P. Zhang, M. Philipose, B. Priyantha, J. Liu, and D. Ganesan, "Glimpse: A programmable early-discard camera architecture for continuous mobile vision," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 292–305.

[126] R. LiKamWa and L. Zhong, "Starfish: Efficient concurrency support for computer vision applications," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 2015, pp. 213–226.

[127] P. Guo and W. Hu, "Potluck: Cross-application approximate deduplication for computation-intensive mobile applications," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 271–284.

[128] K. Boos, D. Chu, and E. Cuervo, "Flashback: Immersive virtual reality on mobile devices via rendering memoization," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 2016, pp. 291–304.

[129] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 99–114.

[130] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 276–286.

[131] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 950–961.

[132] U. Drolia, K. Guo, and P. Narasimhan, "Precog: prefetching for image recognition applications at the edge," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 17.

[133] S. Venugopal, M. Gazzetti, Y. Gkoufas, and K. Katrinis, "Shadow puppets: Cloud-level accurate {AI} inference at the speed and economy of edge," in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[134] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," in *ACM SIGPLAN Notices*, vol. 53, no. 6. ACM, 2018, pp. 31–43.

[135] J. Zhao, R. Mortier, J. Crowcroft, and L. Wang, "Privacy-preserving machine learning based data analytics on edge devices," in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. ACM, 2018, pp. 341–346.

[136] S. S. Ogden and T. Guo, "{MODI}: Mobile deep inference made efficient by edge computing," in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[137] Y. Fang, S. M. Shalmani, and R. Zheng, "Cachenet: A model caching framework for deep learning inference on the edge," *arXiv preprint arXiv:2007.01793*, 2020.

[138] A. Mayberry, P. Hu, B. Marlin, C. Salthouse, and D. Ganesan, "ishadow: design of a wearable, real-time mobile gaze tracker," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, 2014, pp. 82–94.

[139] L. Xu, A. Iyengar, and W. Shi, "Cha: A caching framework for home-based voice assistant systems," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 293–306.

[140] L. Lovagnini, W. Zhang, F. H. Bijarbooneh, and P. Hui, "Circe: Real-time caching for instance recognition on cloud environments and multi-core architectures," in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, pp. 346–354.

[141] Y. Chen, S. Biookaghazadeh, and M. Zhao, "Exploring the capabilities of mobile devices supporting deep learning," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2018, pp. 17–18.

[142] D. Li, T. Salonidis, N. V. Desai, and M. C. Chuah, "Deepcham: Collaborative edge-mediated adaptive deep learning for mobile object recognition," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2016, pp. 64–76.

[143] Y. Huang, Y. Zhu, X. Fan, X. Ma, F. Wang, J. Liu, Z. Wang, and Y. Cui, "Task scheduling with optimized transmission time in collaborative cloud-edge learning," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2018, pp. 1–9.

[144] L. Valerio, A. Passarella, and M. Conti, "A communication efficient distributed learning framework for smart environments," *Pervasive and Mobile Computing*, vol. 41, pp. 46–68, 2017.

[145] T. Xing, S. S. Sandha, B. Balaji, S. Chakraborty, and M. Srivastava, "Enabling edge devices that learn from each other: Cross modal training for activity recognition," in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*. ACM, 2018, pp. 37–42.

[146] O. Valery, P. Liu, and J.-J. Wu, "Cpu/gpu collaboration techniques for transfer learning on mobile devices," in *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2017, pp. 477–484.

[147] ——, "Low precision deep learning training on mobile heterogeneous platform," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 2018, pp. 109–117.

[148] T. Miu, P. Missier, and T. Plötz, "Bootstrapping personalised human activity recognition models using online active learning," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. IEEE, 2015, pp. 1138–1147.

[149] S. Ambrogio, P. Narayanan, H. Tsai, C. Mackin, K. Spoon, A. Chen, A. Fasoli, A. Friz, and G. W. Burr, "Accelerating deep neural networks with analog memory devices," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 149–152.

[150] F. Shahmohammadi, A. Hosseini, C. E. King, and M. Sarrafzadeh, "Smartwatch based activity recognition using active learning," in *Proceedings of the Second IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies*. IEEE Press, 2017, pp. 321–329.

[151] S. Flutura, A. Seiderer, I. Aslan, C.-T. Dang, R. Schwarz, D. Schiller, and E. André, "Drinkwatch: A mobile wellbeing application based on interactive and cooperative machine learning," in *Proceedings of the 2018 International Conference on Digital Health*. ACM, 2018, pp. 65–74.

[152] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.

[153] H. Zeng and V. Prasanna, "Graphact: Accelerating gcn training on cpu-fpga heterogeneous platforms," in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 255–265. [Online]. Available: https://doi.org/10.1145/3373087.3375312

[154] J. Konečnỳ, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," *arXiv preprint arXiv:1511.03575*, 2015.

[155] J. Konečnỳ, H. B. McMahan, X. Y. Felix, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016.

[156] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[157] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[158] S. R. Pandey, N. H. Tran, M. Bennis, Y. K. Tun, A. Manzoor, and C. S. Hong, "A crowdsourcing framework for on-device federated learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 5, pp. 3241–3256, 2020.

[159] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[160] Y. Wang, "Co-op: Cooperative machine learning from mobile devices," 2017.

[161] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning via over-the-air computation," *IEEE Transactions on Wireless Communications*, vol. 19, no. 3, pp. 2022–2035, 2020.

[162] F. Ang, L. Chen, N. Zhao, Y. Chen, W. Wang, and F. R. Yu, "Robust federated learning with noisy communication," *IEEE Transactions on Communications*, pp. 1–1, 2020.

[163] M. M. Amiri and D. Gündüz, "Federated learning over wireless fading channels," *IEEE Transactions on Wireless Communications*, vol. 19, no. 5, pp. 3546–3557, 2020.

[164] S. Savazzi, M. Nicoli, and V. Rampa, "Federated learning with cooperating devices: A consensus approach for massive iot networks," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4641–4654, 2020.

[165] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," 04 2016.

[166] J. Konečnỳ, "Stochastic, distributed and federated optimization for machine learning," *arXiv preprint arXiv:1707.01155*, 2017.

[167] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[168] C. Hardy, E. Le Merrer, and B. Sericola, "Distributed deep learning on edge-devices: feasibility via adaptive compression," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2017, pp. 1–8.

[169] S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar, "Expanding the reach of federated learning by reducing client resource requirements," *arXiv preprint arXiv:1812.07210*, 2018.

[170] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, 2020.

[171] X. Mo and J. Xu, "Energy-efficient federated edge learning with joint communication and computation design," *arXiv preprint arXiv:2003.00199*, 2020.

[172] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 234–243.

[173] S. Tang, W. Zhou, L. Chen, L. Lai, J. Xia, and L. Fan, "Battery-constrained federated edge learning in uav-enabled iot for b5g/6g networks," *arXiv preprint arXiv:2101.12472*, 2021.

[174] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[175] I. Mohammed, S. Tabatabai, A. Al-Fuqaha, F. El Bouanani, J. Qadir, B. Qolomany, and M. Guizani, "Budgeted online selection of candidate iot clients to participate in federated learning," *IEEE Internet of Things Journal*, 2020.

[176] Y. Ye, S. Li, F. Liu, Y. Tang, and W. Hu, "Edgefed: optimized federated learning based on edge computing," *IEEE Access*, vol. 8, pp. 209 191–209 198, 2020.

[177] J. Kang, Z. Xiong, D. Niyato, H. Yu, Y.-C. Liang, and D. I. Kim, "Incentive design for efficient federated learning in mobile networks: A contract theory approach," in *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*. IEEE, 2019, pp. 1–5.

[178] Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, "A learning-based incentive mechanism for federated learning," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6360–6368, 2020.

[179] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[180] M. S. H. Abad, E. Ozfatura, D. Gunduz, and O. Ercetin, "Hierarchical federated learning across heterogeneous cellular networks," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8866–8870.

[181] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu, "Hfel: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 10, pp. 6535–6548, 2020.

[182] N. Mhaisen, A. Awad, A. Mohamed, A. Erbad, and M. Guizani, "Optimal user-edge assignment in hierarchical federated learning based on statistical properties and network topology constraints," *IEEE Transactions on Network Science and Engineering*, 2021.

[183] C. Briggs, Z. Fan, and P. Andras, "Federated learning with hierarchical clustering of local updates to improve training on non-iid data," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–9.

[184] H. Chai, S. Leng, Y. Chen, and K. Zhang, "A hierarchical blockchain-enabled federated learning algorithm for knowledge sharing in internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[185] W. Yang, S. Wang, J. Hu, G. Zheng, J. Yang, and C. Valli, "Securing deep learning based edge finger-vein biometrics with binary decision diagram," *IEEE Transactions on Industrial Informatics*, 2019.

[186] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1175–1191.

[187] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 70–82, 2020.

[188] C. Gentry *et al.*, "Fully homomorphic encryption using ideal lattices." in *Stoc*, vol. 9, no. 2009, 2009, pp. 169–178.

[189] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[190] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2020.

[191] C. Dwork, "Differential privacy," *Encyclopedia of Cryptography and Security*, pp. 338–340, 2011.

[192] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[193] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," *arXiv preprint arXiv:1710.06963*, 2017.

[194] H. H. Zhuo, W. Feng, Q. Xu, Q. Yang, and Y. Lin, "Federated reinforcement learning," *arXiv preprint arXiv:1901.08277*, 2019.

[195] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, and Q. Yang, "Secureboost: A lossless federated learning framework," *arXiv preprint arXiv:1901.08755*, 2019.

[196] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12. Madison, WI, USA: Omnipress, 2012, p. 1467–1474.

[197] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *Advances in neural information processing systems*, 2017, pp. 3517–3529.

[198] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866*, 2018.

[199] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[200] P. Blanchard, R. Guerraoui, J. Stainer *et al.*, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 119–129.

[201] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, p. 44, 2017.

[202] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 5650–5659.

[203] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[204] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2938–2948.

[205] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," *arXiv preprint arXiv:1811.12470*, 2018.

[206] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016, pp. 1050–1059.

[207] S. Yao, Y. Zhao, H. Shao, A. Zhang, C. Zhang, S. Li, and T. Abdelzaher, "Rdeepsense: Reliable deep mobile computing models with uncertainty estimations," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, p. 173, 2018.

[208] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[209] A. Hard, K. Rao, R. Mathews, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[210] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays, "Federated learning of out-of-vocabulary words," *arXiv preprint arXiv:1903.10635*, 2019.

[211] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," *arXiv preprint arXiv:1812.02903*, 2018.

[212] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," *arXiv preprint arXiv:1906.04329*, 2019.

[213] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas, "Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation," in *International MICCAI Brainlesion Workshop*. Springer, 2018, pp. 92–104.

[214] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," *arXiv preprint arXiv:1905.06731*, 2019.

[215] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, 2020.

[216] D. Gao, C. Ju, X. Wei, Y. Liu, T. Chen, and Q. Yang, "Hhhfl: Hierarchical heterogeneous horizontal federated learning for electroencephalography," *arXiv preprint arXiv:1909.05784*, 2019.

[217] W. Zhang, T. Zhou, Q. Lu, X. Wang, C. Zhu, H. Sun, Z. Wang, S. K. Lo, and F.-Y. Wang, "Dynamic fusion-based federated learning for covid-19 detection," *IEEE Internet of Things Journal*, 2021.

[218] A. Qayyum, K. Ahmad, M. A. Ahsan, A. Al-Fuqaha, and J. Qadir, "Collaborative federated learning for healthcare: Multi-modal covid-19 diagnosis at the edge," *arXiv preprint arXiv:2101.07511*, 2021.

[219] A. Vaid, S. K. Jaladanki, J. Xu, S. Teng, A. Kumar, S. Lee, S. Somani, I. Paranjpe, J. K. De Freitas, T. Wanyan *et al.*, "Federated learning of electronic health records to improve mortality prediction in hospitalized patients with covid-19: Machine learning approach," *JMIR medical informatics*, vol. 9, no. 1, p. e24207, 2021.

[220] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," *IEEE Transactions on Communications*, 2019.

[221] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4298–4311, 2020.

[222] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, M. D. Mueck, and S. Srikanteswara, "Energy demand prediction with federated learning for electric vehicle networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[223] T. D. Nguyen, S. Marchal, M. Miettinen, N. Asokan, and A. Sadeghi, "Dïot: A self-learning system for detecting compromised iot devices," *CoRR, vol. abs/1804.07474*, 2018.

[224] S. Chen, Y. Liu, X. Gao, and Z. Han, "Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices," in *Chinese Conference on Biometric Recognition*. Springer, 2018, pp. 428–438.

[225] C. N. Duong, K. G. Quach, N. Le, N. Nguyen, and K. Luu, "Mobiface: A lightweight deep learning face recognition on mobile devices," *arXiv preprint arXiv:1811.11080*, 2018.

[226] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[227] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 2019, pp. 4780–4789.

[228] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang, "Adanet: Adaptive structural learning of artificial neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 874–883.

[229] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[230] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[231] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[232] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[233] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze, "Fastdepth: Fast monocular depth estimation on embedded systems," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6101–6108.

[234] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[235] Z. Qin, Z. Zhang, S. Zhang, H. Yu, and Y. Peng, "Merging-and-evolution networks for mobile vision applications," *IEEE Access*, vol. 6, pp. 31 294–31 306, 2018.

[236] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[237] S. Bhattacharya and N. D. Lane, "From smart to deep: Robust activity recognition on smartwatches using deep learning," in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, 2016, pp. 1–6.

[238] B. Almaslukh, J. Al Muhtadi, and A. M. Artoli, "A robust convolutional neural network for online smartphone-based human activity recognition," *Journal of Intelligent & Fuzzy Systems*, no. Preprint, pp. 1–12, 2018.

[239] B. Almaslukh, A. Artoli, and J. Al-Muhtadi, "A robust deep learning approach for position-independent smartphone-based human activity recognition," *Sensors*, vol. 18, no. 11, p. 3726, 2018.

[240] P. Sundaramoorthy, G. K. Gudur, M. R. Moorthy, R. N. Bhandari, and V. Vijayaraghavan, "Harnet: Towards on-device incremental learning using deep ensembles on constrained devices," in *Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning*. ACM, 2018, pp. 31–36.

[241] V. Radu, N. D. Lane, S. Bhattacharya, C. Mascolo, M. K. Marina, and F. Kawsar, "Towards multimodal deep learning for activity recognition on mobile devices," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*. ACM, 2016, pp. 185–188.

[242] F. Cruciani, I. Cleland, C. Nugent, P. McCullagh, K. Synnes, and J. Hallberg, "Automatic annotation for human activity recognition in free living using a smartphone," *Sensors*, vol. 18, no. 7, p. 2203, 2018.

[243] X. Bo, C. Poellabauer, M. K. O'Brien, C. K. Mummidisetty, and A. Jayaraman, "Detecting label errors in crowd-sourced smartphone sensor data," in *2018 International Workshop on Social Sensing (SocialSens)*. IEEE, 2018, pp. 20–25.

[244] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A unified deep learning framework for time-series mobile sensing data processing," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 351–360.

[245] S. Yao, Y. Zhao, S. Hu, and T. Abdelzaher, "Qualitydeepsense: Quality-aware deep learning framework for internet of things applications with sensor-temporal attention," in *Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning*. ACM, 2018, pp. 42–47.

[246] C. Streiffer, R. Raghavendra, T. Benson, and M. Srivatsa, "Darnet: a deep learning solution for distracted driving detection," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*. ACM, 2017, pp. 22–28.

[247] L. Liu, C. Karatas, H. Li, S. Tan, M. Gruteser, J. Yang, Y. Chen, and R. P. Martin, "Toward detection of unsafe driving with wearables," in *Proceedings of the 2015 workshop on Wearable Systems and Applications*. ACM, 2015, pp. 27–32.

[248] C. Bo, X. Jian, X.-Y. Li, X. Mao, Y. Wang, and F. Li, "You're driving and texting: detecting drivers using personal smart phones by leveraging inertial sensors," in *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM, 2013, pp. 199–202.

[249] J. Yang, S. Sidhom, G. Chandrasekaran, T. Vu, H. Liu, N. Cecan, Y. Chen, M. Gruteser, and R. P. Martin, "Detecting driver phone use leveraging car speakers," in *Proceedings of the 17th annual international conference on Mobile computing and networking*. ACM, 2011, pp. 97–108.

[250] N. D. Lane, P. Georgiev, and L. Qendro, "Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2015, pp. 283–294.

[251] P. Georgiev, S. Bhattacharya, N. D. Lane, and C. Mascolo, "Low-resource multi-task audio sensing for mobile and embedded devices via shared deep neural network representations," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, p. 50, 2017.

[252] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[253] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, 2014, pp. 1269–1277.

[254] P. Maji, D. Bates, A. Chadwick, and R. Mullins, "Adapt: optimizing cnn inference on iot and mobile devices using approximately separable 1-d kernels," in *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*. ACM, 2017, p. 43.

[255] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[256] P. Wang and J. Cheng, "Accelerating convolutional neural networks for mobile applications," in *Proceedings of the 24th ACM international conference on Multimedia*. ACM, 2016, pp. 541–545.

[257] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, 2016, pp. 176–189.

[258] C. Buciluă, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 535–541.

[259] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[260] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[261] N. Komodakis and S. Zagoruyko, "Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer," in *ICLR*, Paris, France, Jun. 2017. [Online]. Available: https://hal-enpc.archives-ouvertes.fr/hal-01832769

[262] B. B. Sau and V. N. Balasubramanian, "Deep model compression: Distilling knowledge from noisy teachers," *arXiv preprint arXiv:1610.09650*, 2016.

[263] E. J. Crowley, G. Gray, and A. J. Storkey, "Moonshine: Distilling with cheap convolutions," in *Advances in Neural Information Processing Systems*, 2018, pp. 2888–2898.

[264] D. Li, X. Wang, and D. Kong, "Deeprebirth: Accelerating deep neural network execution on mobile devices," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[265] G. Zhou, Y. Fan, R. Cui, W. Bian, X. Zhu, and K. Gai, "Rocket launching: A universal and efficient framework for training well-

performing light net," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[266] R. G. Lopes, S. Fenu, and T. Starner, "Data-free knowledge distillation for deep neural networks," *arXiv preprint arXiv:1710.07535*, 2017.

[267] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[268] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[269] A. Wong, M. Famuori, M. J. Shafiee, F. Li, B. Chwyl, and J. Chung, "Yolo nano: a highly compact you only look once convolutional neural network for object detection," 2019.

[270] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[271] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[272] M. J. Shafiee, F. Li, B. Chwyl, and A. Wong, "Squishednets: Squishing squeezenet further for edge device scenarios via deep evolutionary synthesis," *arXiv preprint arXiv:1711.07459*, 2017.

[273] K. Yang, T. Xing, Y. Liu, Z. Li, X. Gong, X. Chen, and D. Fang, "Cdeeparch: a compact deep neural network architecture for mobile sensing," *IEEE/ACM Transactions on Networking*, 2019.

[274] J. Zhang, X. Wang, D. Li, and Y. Wang, "Dynamically hierarchy revolution: dirnet for compressing recurrent neural network on mobile devices," *arXiv preprint arXiv:1806.01248*, 2018.

[275] Y. Shen, T. Han, Q. Yang, X. Yang, Y. Wang, F. Li, and H. Wen, "Cs-cnn: Enabling robust and efficient convolutional neural networks inference for internet-of-things applications," *IEEE Access*, vol. 6, pp. 13 439–13 448, 2018.

[276] J. Guo and M. Potkonjak, "Pruning filters and classes: Towards on-device customization of convolutional neural networks," in *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications*. ACM, 2017, pp. 13–17.

[277] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[278] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1586–1595.

[279] F. Manessi, A. Rozza, S. Bianco, P. Napoletano, and R. Schettini, "Automated pruning for deep neural network compression," in *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 2018, pp. 657–664.

[280] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *arXiv preprint arXiv:1611.06440*, vol. 3, 2016.

[281] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 2133–2144.

[282] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5687–5695.

[283] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 2017, p. 4.

[284] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[285] C.-F. Chen, G. G. Lee, V. Sritapan, and C.-Y. Lin, "Deep convolutional neural network on ios mobile devices," in *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2016, pp. 130–135.

[286] J.-H. Luo and J. Wu, "Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference," *Pattern Recognition*, p. 107461, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320320302648

[287] J. Guo, W. Zhang, W. Ouyang, and D. Xu, "Model compression using progressive channel pruning," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2020.

[288] O. Oyedotun, D. Aouada, and B. Ottersten, "Structured compression of deep neural networks with debiased elastic group lasso," in *The IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[289] P. Singh, V. K. Verma, P. Rai, and V. Namboodiri, "Leveraging filter correlations for deep model compression," in *The IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[290] J. Wang, H. Bai, J. Wu, and J. Cheng, "Bayesian automatic model compression," *IEEE Journal of Selected Topics in Signal Processing*, pp. 1–1, 2020.

[291] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[292] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[293] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[294] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[295] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[296] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[297] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[298] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *arXiv preprint arXiv:1510.03009*, 2015.

[299] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," 2011.

[300] R. Alvarez, R. Prabhavalkar, and A. Bakhtin, "On the efficient representation and execution of deep acoustic models," *arXiv preprint arXiv:1607.04683*, 2016.

[301] M. A. Nasution, D. Chahyati, and M. I. Fanany, "Faster r-cnn with structured sparsity learning and ristretto for mobile environment," in *2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. IEEE, 2017, pp. 309–314.

[302] P. Peng, Y. Mingyu, and X. Weisheng, "Running 8-bit dynamic fixed-point convolutional neural network on low-cost arm platforms," in *2017 Chinese Automation Congress (CAC)*. IEEE, 2017, pp. 4564–4568.

[303] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1131–1135.

[304] S. H. F. Langroudi, T. Pandit, and D. Kudithipudi, "Deep learning inference on embedded devices: Fixed-point vs posit," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE, 2018, pp. 19–23.

[305] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Advances in Neural Information Processing Systems*, 2014, pp. 963–971.

[306] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, 2015, pp. 1117–1125.

[307] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 68–81.

[308] X. Zeng, K. Cao, and M. Zhang, "Mobiledeeppill: A small-footprint mobile deep learning system for recognizing unconstrained pill im-

ages," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 56–67.

[309] P. Wang, Q. Hu, Z. Fang, C. Zhao, and J. Cheng, "Deepsearch: A fast image search framework for mobile devices," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 14, no. 1, p. 6, 2018.

[310] B. Kim, Y. Jeon, H. Park, D. Han, and Y. Baek, "Design and implementation of the vehicular camera system using deep neural network compression," in *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications*. ACM, 2017, pp. 25–30.

[311] X. Xu, S. Yin, and P. Ouyang, "Fast and low-power behavior analysis on vehicles using smartphones," in *2017 6th International Symposium on Next Generation Electronics (ISNE)*. IEEE, 2017, pp. 1–4.

[312] B. Wang, F. Ma, L. Ge, H. Ma, H. Wang, and M. A. Mohamed, "Icing-edgenet: a pruning lightweight edge intelligent method of discriminative driving channel for ice thickness of transmission lines," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–12, 2020.

[313] F. Shang, J. Lai, J. Chen, W. Xia, and H. Liu, "A model compression based framework for electrical equipment intelligent inspection on edge computing environment," in *2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. IEEE, 2021, pp. 406–410.

[314] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "Rstensorflow: Gpu enabled tensorflow for deep learning on commodity android devices," in *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications*. ACM, 2017, pp. 7–12.

[315] M. Loukadakis, J. Cano, and M. O'Boyle, "Accelerating deep neural networks on low power heterogeneous architectures," 2018.

[316] S. S. L. Oskouei, H. Golestani, M. Kachuee, M. Hashemi, H. Mohammadzade, and S. Ghiasi, "Gpu-based acceleration of deep convolutional neural networks on mobile platforms," *Distrib. Parallel Clust. Comput*, 2015.

[317] S. S. Latifi Oskouei, H. Golestani, M. Hashemi, and S. Ghiasi, "Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android," in *Proceedings of the 24th ACM international conference on Multimedia*. ACM, 2016, pp. 1201–1205.

[318] P.-K. Tsung, S.-F. Tsai, A. Pai, S.-J. Lai, and C. Lu, "High performance deep neural network on low cost mobile gpu," in *2016 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2016, pp. 69–70.

[319] S. Rizvi, G. Cabodi, D. Patti, and G. Francini, "Gpgpu accelerated deep object classification on a heterogeneous mobile platform," *Electronics*, vol. 5, no. 4, p. 88, 2016.

[320] S. Rizvi, D. Patti, T. Björklund, G. Cabodi, and G. Francini, "Deep classifiers-based license plate detection, localization and recognition on gpu-powered mobile platform," *Future Internet*, vol. 9, no. 4, p. 66, 2017.

[321] S. Rizvi, G. Cabodi, and G. Francini, "Optimized deep neural networks for real-time object classification on embedded gpus," *Applied Sciences*, vol. 7, no. 8, p. 826, 2017.

[322] S. Rizvi, G. Cabodi, D. Patti, and M. Gulzar, "A general-purpose graphics processing unit (gpgpu)-accelerated robotic controller using a low power mobile platform," *Journal of Low Power Electronics and Applications*, vol. 7, no. 2, p. 10, 2017.

[323] Q. Cao, N. Balasubramanian, and A. Balasubramanian, "Mobirnn: Efficient recurrent neural network execution on mobile gpu," in *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications*. ACM, 2017, pp. 1–6.

[324] H. Guihot, "Renderscript," in *Pro Android Apps Performance Optimization*. Springer, 2012, pp. 231–263.

[325] M. Motamedi, D. Fong, and S. Ghiasi, "Fast and energy-efficient cnn inference on iot devices," *arXiv preprint arXiv:1611.07151*, 2016.

[326] ——, "Cappuccino: Efficient cnn inference software synthesis for mobile system-on-chips," *IEEE Embedded Systems Letters*, vol. 11, no. 1, pp. 9–12, 2018.

[327] ——, "Machine intelligence on resource-constrained iot devices: The case of thread granularity optimization for cnn inference," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 151, 2017.

[328] L. N. Huynh, R. K. Balan, and Y. Lee, "Deepsense: A gpu-based deep convolutional neural network framework on commodity mobile devices," in *Proceedings of the 2016 Workshop on Wearable Systems and Applications*. ACM, 2016, pp. 25–30.

[329] B. Taylor, V. S. Marco, and Z. Wang, "Adaptive optimization for opencl programs on embedded heterogeneous systems," in *ACM SIGPLAN Notices*, vol. 52, no. 5. ACM, 2017, pp. 11–20.

[330] S. Rallapalli, H. Qiu, A. Bency, S. Karthikeyan, R. Govindan, B. Manjunath, and R. Urgaonkar, "Are very deep neural networks feasible on mobile devices," *IEEE Trans. Circ. Syst. Video Technol*, 2016.

[331] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[332] M. Bettoni, G. Urgese, Y. Kobayashi, E. Macii, and A. Acquaviva, "A convolutional neural network fully implemented on fpga for embedded platforms," in *2017 New Generation of CAS (NGCAS)*. IEEE, 2017, pp. 49–52.

[333] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 45–54.

[334] S.-S. Park, K.-B. Park, and K.-S. Chung, "Implementation of a cnn accelerator on an embedded soc platform using sdsoc," in *Proceedings of the 2nd International Conference on Digital Signal Processing*. ACM, 2018, pp. 161–165.

[335] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Understanding the limitations of existing energy-efficient design approaches for deep neural networks," *Energy*, vol. 2, no. L1, p. L3, 2018.

[336] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[337] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[338] Z. Liu, P. N. Whatmough, and M. Mattina, "Systolic tensor array: An efficient structured-sparse gemm accelerator for mobile cnn inference," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 34–37, 2020.

[339] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2016, pp. 123–136.

[340] P. Georgiev, N. D. Lane, C. Mascolo, and D. Chu, "Accelerating mobile audio sensing algorithms through on-chip gpu offloading," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 306–318.

[341] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 2016, p. 23.

[342] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "Accelerated deep learning inference for embedded and wearable devices using deepx," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion*. ACM, 2016, pp. 109–109.

[343] N. D. Lane, S. Bhattacharya, A. Mathur, C. Forlivesi, and F. Kawsar, "Dxtk: Enabling resource-efficient deep learning on mobile and embedded devices with the deepx toolkit." in *MobiCASE*, 2016, pp. 98–107.

[344] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300.

[345] C. Ma, Z. Zhu, J. Ye, J. Yang, J. Pei, S. Xu, R. Zhou, C. Yu, F. Mo, B. Wen *et al.*, "Deeprt: deep learning for peptide retention time prediction in proteomics," *arXiv preprint arXiv:1705.05368*, 2017.

[346] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin, "Accelerating convolutional neural network with fft on embedded hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 9, pp. 1737–1749, 2018.

[347] 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, 2018.

[348] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *2017 IEEE 25th international conference on network protocols (ICNP)*. IEEE, 2017, pp. 1–2.

[349] A. E. Eshratifar and M. Pedram, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. ACM, 2018, pp. 111–116.

[350] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1. ACM, 2017, pp. 615–629.

[351] S. A. Osia, A. S. Shamsabadi, S. Sajadmanesh, A. Taheri, K. Katevas, H. R. Rabiee, N. D. Lane, and H. Haddadi, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *IEEE Internet of Things Journal*, 2020.

[352] C. Liu, S. Chakraborty, and P. Mittal, "Deeprotect: Enabling inference-based access control on mobile sensing applications," *arXiv preprint arXiv:1702.06159*, 2017.

[353] C. Xu, J. Ren, L. She, Y. Zhang, Z. Qin, and K. Ren, "Edgesanitizer: Locally differentially private deep inference at the edge for mobile data analytics," *IEEE Internet of Things Journal*, 2019.

[354] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *computer*, vol. 50, no. 10, pp. 58–67, 2017.

[355] M. Ali, A. Anjum, 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)*. IEEE, 2018, pp. 1–10.

[356] P. Sanabria, J. I. Benedetto, A. Neyem, J. Navon, and C. Poellabauer, "Code offloading solutions for audio processing in mobile healthcare applications: a case study," in *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2018, pp. 117–121.

[357] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo, and A. Ylä-Jääski, "Latency and throughput characterization of convolutional neural networks for mobile computer vision," in *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, 2018, pp. 204–215.

[358] B. Qi, M. Wu, and L. Zhang, "A dnn-based object detection system on mobile cloud computing," in *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*. IEEE, 2017, pp. 1–6.

[359] X. Ran, H. Chen, Z. Liu, and J. Chen, "Delivering deep learning to mobile devices via offloading," in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 2017, pp. 42–47.

[360] 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, 2018, pp. 1421–1429.

[361] P. Georgiev, N. D. Lane, K. K. Rachuri, and C. Mascolo, "Leo: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 2016, pp. 320–333.

[362] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 43–56.

[363] C. Streiffer, A. Srivastava, V. Orlikowski, Y. Velasco, V. Martin, N. Raval, A. Machanavajjhala, and L. P. Cox, "eprivateeye: To the edge and beyond!" in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 18.

[364] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*. ACM, 2018, pp. 31–36.

[365] 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," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2018, pp. 1–6.

[366] Y. Tian, J. Yuan, S. Yu, and Y. Hou, "Lep-cnn: A lightweight edge device assisted privacy-preserving cnn inference solution for iot," *arXiv preprint arXiv:1901.04100*, 2019.

[367] C. Zhang and Z. Zheng, "Task migration for mobile edge computing using deep reinforcement learning," *Future Generation Computer Systems*, vol. 96, pp. 111–118, 2019.

[368] H.-J. Jeong, I. Jeong, H.-J. Lee, and S.-M. Moon, "Computation offloading for machine learning web apps in the edge server environment," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1492–1499.

[369] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 15.

[370] M. Xu, F. Qian, and S. Pushp, "Enabling cooperative inference of deep learning on wearables and smartphones," *arXiv preprint arXiv:1712.03073*, 2017.

[371] P. Liu, B. Qi, and S. Banerjee, "Edgeeye: An edge service framework for real-time intelligent video analytics," in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*. ACM, 2018, pp. 1–6.

[372] M. Song, K. Zhong, J. Zhang, Y. Hu, D. Liu, W. Zhang, J. Wang, and T. Li, "In-situ ai: Towards autonomous and incremental deep learning for iot systems," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 92–103.

[373] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Musical chair: Efficient real-time recognition using collaborative iot devices," *arXiv preprint arXiv:1802.02138*, 2018.

[374] N. Talagala, S. Sundararaman, V. Sridhar, D. Arteaga, Q. Luo, S. Subramanian, S. Ghanta, L. Khermosh, and D. Roselli, "{ECO}: Harmonizing edge and cloud with ml/dl orchestration," in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[375] E. De Coninck, S. Bohez, S. Leroux, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "Dianne: a modular framework for designing, training and deploying deep neural networks on heterogeneous distributed infrastructure," *Journal of Systems and Software*, vol. 141, pp. 52–65, 2018.

[376] Y. Fukushima, D. Miura, T. Hamatani, H. Yamaguchi, and T. Higashino, "Microdeep: In-network deep learning by micro-sensor coordination for pervasive computing," in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2018, pp. 163–170.

[377] T. Bach, M. A. Tariq, R. Mayer, and K. Rothermel, "Knowledge is at the edge! how to search in distributed machine learning models," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2017, pp. 410–428.

[378] A. Morshed, P. P. Jayaraman, T. Sellis, D. Georgakopoulos, M. Villari, and R. Ranjan, "Deep osmosis: Holistic distributed deep learning in osmotic computing," *IEEE Cloud Computing*, vol. 4, no. 6, pp. 22–32, 2017.

[379] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 328–339.

[380] A. Yousefpour, S. Devic, B. Q. Nguyen, A. Kreidieh, A. Liao, A. M. Bayen, and J. P. Jue, "Guardians of the deep fog: Failure-resilient dnn inference from edge to cloud," in *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, 2019, pp. 25–31.

[381] N. Muslim, S. Islam, and J.-C. Grégoire, "Offloading framework for computation service in the edge cloud and core cloud: A case study for face recognition," *International Journal of Network Management*, vol. 31, no. 4, p. e2146, 2021.

[382] A. Ferdowsi, U. Challita, and W. Saad, "Deep learning for reliable mobile edge analytics in intelligent transportation systems: An overview," *ieee vehicular technology magazine*, vol. 14, no. 1, pp. 62–70, 2019.

[383] L. Li, K. Ota, and M. Dong, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4665–4673, 2018.

[384] B. Tang, Z. Chen, G. Hefferman, S. Pei, 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.

[385] C. Liu, Y. Cao, Y. Luo, G. Chen, V. Vokkarane, M. Yunsheng, S. Chen, and P. Hou, "A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 249–261, 2017.

[386] B. Javadi, Q. L. Trieu, K. M. Matawie, and R. N. Calheiros, "Smart food scanner system based on mobile edge computing," in *2020 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2020, pp. 20–27.

[387] T. Muhammed, R. Mehmood, A. Albeshri, and I. Katib, "Ubehealth: a personalized ubiquitous cloud and edge-enabled networked healthcare system for smart cities," *IEEE Access*, vol. 6, pp. 32 258–32 285, 2018.

[388] M. Schwabacher and K. Goebel, "A survey of artificial intelligence for prognostics." in *AAAI Fall Symposium: Artificial Intelligence for Prognostics*, 2007, pp. 108–115.

[389] A. He, K. K. Bae, T. R. Newman, J. Gaeddert, K. Kim, R. Menon, L. Morales-Tirado, Y. Zhao, J. H. Reed, W. H. Tranter *et al.*, "A survey of artificial intelligence for cognitive radios," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 4, pp. 1578–1592, 2010.

[390] A. Bahrammirzaee, "A comparative survey of artificial intelligence applications in finance: artificial neural networks, expert system and hybrid intelligent systems," *Neural Computing and Applications*, vol. 19, no. 8, pp. 1165–1195, 2010.

[391] Y. Zhang, J. Ren, J. Liu, C. Xu, H. Guo, and Y. Liu, "A survey on emerging computing paradigms for big data," *Chinese Journal of Electronics*, vol. 26, no. 1, pp. 1–12, 2017.

[392] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of big data*, vol. 2, no. 1, p. 8, 2015.

[393] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 workshop on mobile big data*, 2015, pp. 37–42.

[394] T.-h. Kim, C. Ramos, and S. Mohammed, "Smart city and iot," 2017.

[395] K. Su, J. Li, and H. Fu, "Smart city and the applications," in *2011 international conference on electronics, communications and control (ICECC)*. IEEE, 2011, pp. 1028–1031.

[396] W. Zhang, B. Han, and P. Hui, "Jaguar: Low latency mobile augmented reality with flexible tracking," in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, pp. 355–363.

[397] W. Zhang, S. Lin, F. H. Bijarbooneh, H. F. Cheng, and P. Hui, "Cloudar: A cloud-based framework for mobile augmented reality," in *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, 2017, pp. 194–200.

[398] S. Lin, H. F. Cheng, W. Li, Z. Huang, P. Hui, and C. Peylo, "Ubii: Physical world interaction through augmented reality," *IEEE Transactions on Mobile Computing*, vol. 16, no. 3, pp. 872–885, 2016.

[399] J. McCarthy, "What is artificial intelligence?" 1998.

[400] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," 2002.

[401] S. Dick, "Artificial intelligence," 2019.

[402] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[403] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, "An introduction to decision tree modeling," *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 18, no. 6, pp. 275–285, 2004.

[404] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?" in *International workshop on machine learning and data mining in pattern recognition*. Springer, 2012, pp. 154–168.

[405] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[406] O. Kramer, "K-nearest neighbors," in *Dimensionality reduction with unsupervised nearest neighbors*. Springer, 2013, pp. 13–23.

[407] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[408] G. Hamerly and C. Elkan, "Learning the k in k-means," *Advances in neural information processing systems*, vol. 16, pp. 281–288, 2004.

[409] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[410] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[411] A. Hyvärinen, "Survey on independent component analysis," 1999.

[412] C. C. Paige and M. A. Saunders, "Towards a generalized singular value decomposition," *SIAM Journal on Numerical Analysis*, vol. 18, no. 3, pp. 398–405, 1981.

[413] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[414] H. Hasselt, "Double q-learning," *Advances in neural information processing systems*, vol. 23, pp. 2613–2621, 2010.

[415] N. Sprague and D. Ballard, "Multiple-goal reinforcement learning with modular sarsa (0)," 2003.

[416] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," *arXiv preprint arXiv:1602.04621*, 2016.

[417] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.

[418] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*. Citeseer, 2000, pp. 1008–1014.

[419] M. Nielsen, "A visual proof that neural nets can compute any function," in *Artificial Neural Networks and Deep Learning*. Determination Press, 2016.

[420] D.-X. Zhou, "Universality of deep convolutional neural networks," *Applied and computational harmonic analysis*, vol. 48, no. 2, pp. 787–794, 2020.

[421] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[422] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[423] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[424] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[425] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[426] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[427] G. S. Paschos, G. Iosifidis, M. Tao, D. Towsley, and G. Caire, "The role of caching in future communication systems and networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1111–1125, 2018.

[428] L. Li, G. Zhao, and R. S. Blum, "A survey of caching techniques in cellular networks: Research issues and challenges in content placement and delivery strategies," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 1710–1732, 2018.

[429] T. Li, T. Braud, Y. Li, and P. Hui, "Lifecycle-aware online video caching," *IEEE Transactions on Mobile Computing*, 2020.

[430] B. Jan, H. Farman, M. Khan, M. Imran, I. U. Islam, A. Ahmad, S. Ali, and G. Jeon, "Deep learning in big data analytics: a comparative study," *Computers & Electrical Engineering*, vol. 75, pp. 275–287, 2019.

[431] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[432] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review," *International Journal of Automation and Computing*, vol. 14, no. 5, pp. 503–519, 2017.

[433] X.-W. Chen and X. Lin, "Big data deep learning: challenges and perspectives," *IEEE access*, vol. 2, pp. 514–525, 2014.

[434] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the computational cost of deep learning models," in *2018 IEEE international conference on big data (Big Data)*. IEEE, 2018, pp. 3873–3882.

[435] M. Polese, R. Jana, V. Kounev, K. Zhang, S. Deb, and M. Zorzi, "Machine learning at the edge: A data-driven architecture with applications to 5g cellular networks," *IEEE Transactions on Mobile Computing*, 2020.

[436] M. Aazam and E.-N. Huh, "Dynamic resource provisioning through fog micro datacenter," in *2015 IEEE international conference on pervasive computing and communication workshops (PerCom workshops)*. IEEE, 2015, pp. 105–110.

[437] ——, "Fog computing micro datacenter based dynamic resource estimation and pricing model for iot," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. IEEE, 2015, pp. 687–694.

[438] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5g systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[439] M. Gregori, J. Gómez-Vilardebó, J. Matamoros, and D. Gündüz, "Wireless content caching for small cell and d2d networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1222–1234, 2016.

[440] S. Li, J. Xu, M. Van Der Schaar, and W. Li, "Popularity-driven content caching," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[441] Y. Liu, Q. He, D. Zheng, X. Xia, F. Chen, and B. Zhang, "Data caching optimization in the edge computing environment," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[442] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2021.

[443] A. Trivedi, L. Wang, H. Bal, and A. Iosup, "Sharing and caring of data at the edge," in *3rd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 20)*, 2020.

[444] S. Jeong, S. Kim, and J. Kim, "City data hub: Implementation of standard-based smart city data platform for interoperability," *Sensors*, vol. 20, no. 23, p. 7000, 2020.

[445] H. Soliman, K. Sudan, and A. Mishra, "A smart forest-fire early detection sensory system: Another approach of utilizing wireless sensor and neural networks," in *SENSORS, 2010 IEEE*. IEEE, 2010, pp. 1900–1904.

[446] A. P. Mathur and N. O. Tippenhauer, "Swat: A water treatment testbed for research and training on ics security," in *2016 international workshop on cyber-physical systems for smart water networks (CySWater)*. IEEE, 2016, pp. 31–36.

[447] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, "Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 703–716.

[448] C. M. Ahmed, M. Ochoa, J. Zhou, A. P. Mathur, R. Qadeer, C. Murguia, and J. Ruths, "Noiseprint: Attack detection using sensor and process noise fingerprint in cyber physical systems," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 483–497.

[449] W. Aoudi, M. Iturbe, and M. Almgren, "Truth will out: Departure-based process-level detection of stealthy attacks on control systems," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 817–831.

[450] M. Usman, M. A. Jan, A. Jolfaei, M. Xu, X. He, and J. Chen, "A distributed and anonymous data collection framework based on multilevel edge computing architecture," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6114–6123, 2019.

[451] D. Xu, H. Wang, Y. Li, S. Tarkoma, D. Jin, and P. Hui, "Iot vs. human: A comparison of mobility," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[452] T. Zhang, H. Li, L. Xu, J. Gao, J. Guan, and X. Cheng, "Comprehensive iot sim card anomaly detection algorithm based on big data," in *2019 IEEE International Conferences on Ubiquitous Computing Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS)*, 2019, pp. 602–606.

[453] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[454] M. H. Cheung, F. Hou, and J. Huang, "Delay-sensitive mobile crowdsensing: Algorithm design and economics," *IEEE Transactions on Mobile Computing*, vol. 17, no. 12, pp. 2761–2774, 2018.

[455] Z. Mottaghinia and A. Ghaffari, "Fuzzy logic based distance and energy-aware routing protocol in delay-tolerant mobile sensor networks," *Wireless Personal Communications*, vol. 100, no. 3, pp. 957–976, 2018.

[456] D. M. Toma, T. O'Reilly, J. del Rio, K. Headley, A. Manuel, A. Bröring, and D. Edgington, "Smart sensors for interoperable smart ocean environment," in *OCEANS 2011 IEEE-Spain*. IEEE, 2011, pp. 1–4.

[457] 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.

[458] J. Haladjian, Z. Hodaie, S. Nüske, and B. Brügge, "Gait anomaly detection in dairy cattle," in *Proceedings of the Fourth International Conference on Animal-Computer Interaction*, 2017, pp. 1–8.

[459] N. Wagner, V. Antoine, M.-M. Mialon, R. Lardy, M. Silberberg, J. Koko, and I. Veissier, "Machine learning to detect behavioural anomalies in dairy cows under subacute ruminal acidosis," *Computers and Electronics in Agriculture*, vol. 170, p. 105233, 2020.

[460] S. A. Shammi and Q. Meng, "Use time series ndvi and evi to develop dynamic crop growth metrics for yield modeling," *Ecological Indicators*, vol. 121, p. 107124, 2021.

[461] A. Shah, A. Dubey, V. Hemnani, D. Gala, and D. Kalbande, "Smart farming system: Crop yield prediction using regression techniques," in *Proceedings of International Conference on Wireless Communication*. Springer, 2018, pp. 49–56.

[462] N. G. Rezk, E. E.-D. Hemdan, A.-F. Attia, A. El-Sayed, and M. A. El-Rashidy, "An efficient iot based smart farming system using machine learning algorithms," *Multimedia Tools and Applications*, vol. 80, no. 1, pp. 773–797, 2021.

[463] M. F. Labib, A. S. Rifat, M. M. Hossain, A. K. Das, and F. Nawrine, "Road accident analysis and prediction of accident severity by using machine learning in bangladesh," in *2019 7th International Conference on Smart Computing & Communications (ICSCC)*. IEEE, 2019, pp. 1–5.

[464] Google Glass, *https://en.wikipedia.org/wiki/Google_Glass*, 2019.

[465] Microsoft Hololens, *https://en.wikipedia.org/wiki/Microsoft_HoloLens*, 2019.

[466] T. Braud, P. Zhou, J. Kangasharju, and P. Hui, "Multipath computation offloading for mobile augmented reality," in *In Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom 2020), Austin USA*, 2020.

[467] S. Nakandala, K. Nagrecha, A. Kumar, and Y. Papakonstantinou, "Incremental and approximate computations for accelerating deep cnn inference," *ACM Transactions on Database Systems (TODS)*, vol. 45, no. 4, pp. 1–42, 2020.

[468] X. Liu, H. Li, X. Lu, T. Xie, Q. Mei, F. Feng, and H. Mei, "Understanding diverse usage patterns from large-scale appstore-service profiles," *IEEE Transactions on Software Engineering*, vol. 44, no. 4, pp. 384–411, 2017.

[469] S. Kim, J. Lee, S. Kang, J. Lee, and H.-J. Yoo, "A power-efficient cnn accelerator with similar feature skipping for face recognition in mobile devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 4, pp. 1181–1193, 2020.

[470] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. Ieee, 2005, pp. 886–893.

[471] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[472] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[473] G. Nebehay and R. Pflugfelder, "Consensus-based matching and tracking of keypoints for object tracking," in *IEEE Winter Conference on Applications of Computer Vision*. IEEE, 2014, pp. 862–869.

[474] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, 2012, pp. 97–108.

[475] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 187–198.

[476] M. Graf, C. Timmerer, and C. Mueller, "Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, 2017, pp. 261–271.

[477] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. De Turck, "An http/2-based adaptive streaming framework for 360 virtual reality videos," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 306–314.

[478] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[479] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[480] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[481] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[482] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza, "Power to the people: The role of humans in interactive machine learning," *Ai Magazine*, vol. 35, no. 4, pp. 105–120, 2014.

[483] M. Ware, E. Frank, G. Holmes, M. Hall, and I. H. Witten, "Interactive machine learning: letting users build classifiers," *International Journal of Human-Computer Studies*, vol. 55, no. 3, pp. 281–292, 2001.

[484] J. B. Predd, S. B. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 23, no. 4, pp. 56–69, 2006.

[485] M. Lavassani, S. Forsström, U. Jennehag, and T. Zhang, "Combining fog computing with sensor mote machine learning for industrial iot," *Sensors*, vol. 18, no. 5, p. 1532, 2018.

[486] C. Ma, J. Li, M. Ding, H. H. Yang, F. Shu, T. Q. S. Quek, and H. V. Poor, "On safeguarding privacy and security in the framework of federated learning," *IEEE Network*, pp. 1–7, 2020.

[487] S. Yao, Y. Zhao, H. Shao, C. Zhang, A. Zhang, D. Liu, S. Liu, L. Su, and T. Abdelzaher, "Apdeepsense: Deep learning uncertainty estimation without the pain for iot applications," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 334–343.

[488] Y. Li, X. Tao, X. Zhang, J. Liu, and J. Xu, "Privacy-preserved federated learning for autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[489] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan, "Communication-efficient distributed dual coordinate ascent," in *Advances in neural information processing systems*, 2014, pp. 3068–3076.

[490] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[491] D. Kong, "Science driven innovations powering mobile product: Cloud ai vs. device ai solutions on smart device," *arXiv preprint arXiv:1711.07580*, 2017.

[492] T. Guo, "Cloud-based or on-device: An empirical study of mobile deep inference," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 184–190.

[493] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.

[494] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," *arXiv preprint arXiv:1905.01392*, 2019.

[495] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[496] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[497] B. Fan, X. Liu, X. Su, J. Niu, and P. Hui, "Emgauth: An emg-based smartphone unlocking system using siamese network," in *In Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom 2020), Austin USA*. IEEE, 2020.

[498] D. Wen, H. Han, and A. K. Jain, "Face spoof detection with image distortion analysis," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 4, pp. 746–761, 2015.

[499] T. Plötz and Y. Guan, "Deep learning for human activity recognition in mobile computing," *Computer*, vol. 51, no. 5, pp. 50–59, 2018.

[500] Y. Guan and T. Plötz, "Ensembles of deep lstm learners for activity recognition using wearables," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 2, p. 11, 2017.

[501] M. Shoaib, O. D. Incel, H. Scolten, and P. Havinga, "Resource consumption analysis of online activity recognition on mobile phones and smartwatches," in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2017, pp. 1–6.

[502] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen, "Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2015, pp. 127–140.

[503] A. Rosenfeld and J. K. Tsotsos, "Incremental learning through deep adaptation," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[504] W. T. Ang, P. K. Khosla, and C. N. Riviere, "Nonlinear regression model of a low-*g* mems accelerometer," *IEEE Sensors Journal*, vol. 7, no. 1, pp. 81–88, 2007.

[505] S. G. Klauer, F. Guo, B. G. Simons-Morton, M. C. Ouimet, S. E. Lee, and T. A. Dingus, "Distracted driving and risk of road crashes among novice and experienced drivers," *New England journal of medicine*, vol. 370, no. 1, pp. 54–59, 2014.

[506] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[507] M. Rabbi, S. Ali, T. Choudhury, and E. Berke, "Passive and in-situ assessment of mental and physical well-being using mobile sensors," in *Proceedings of the 13th international conference on Ubiquitous computing*. ACM, 2011, pp. 385–394.

[508] A. Gebhart, "Google home to the amazon echo:'anything you can do...'," *cnet, May*, vol. 18, p. 7, 2017.

[509] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, no. 4, p. 216, 2018.

[510] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[511] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 535–541.

[512] R. G. Baraniuk, "Compressive sensing," *IEEE signal processing magazine*, vol. 24, no. 4, 2007.

[513] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.

[514] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in neural information processing systems*, 1993, pp. 164–171.

[515] J. Van Leeuwen, "On the construction of huffman trees." in *ICALP*, 1976, pp. 382–410.

[516] S. Malki and L. Spaanenburg, "Cnn image processing on a xilinx virtex-ii 6000," in *Proceedings ECCTD*, vol. 3, 2003, pp. 261–264.

[517] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, 2017.

[518] R. Morris, "Tapered floating point: A new floating-point representation," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1578–1579, 1971.

[519] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[520] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[521] B. Blanco-Filgueira, D. García-Lesta, M. Fernández-Sanjurjo, V. M. Brea, and P. López, "Deep learning-based multiple object visual tracking on embedded system for iot and mobile edge computing applications," *IEEE Internet of Things Journal*, 2019.

[522] J. Appleyard, T. Kocisky, and P. Blunsom, "Optimizing performance of recurrent neural networks on gpus," *arXiv preprint arXiv:1604.01946*, 2016.

[523] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2018, pp. 389–400.

[524] S. Liu, Q. Wang, and G. Liu, "A versatile method of discrete convolution and fft (dc-fft) for contact analyses," *Wear*, vol. 243, no. 1-2, pp. 101–111, 2000.

[525] V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib, "Optimizing deep learning inference on embedded systems through adaptive model selection," *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 1, Feb. 2020. [Online]. Available: https://doi.org/10.1145/3371154

[526] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "Pulp-nn: accelerating quantized neural networks on parallel ultra-low-power risc-v processors," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, p. 20190155, 2020.

[527] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[528] K. A. Shatilov, D. Chatzopoulos, A. W. T. Hang, and P. Hui, "Using deep learning and mobile offloading to control a 3d-printed prosthetic hand," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 3, pp. 1–19, 2019.

[529] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 proceedings IEEE Infocom*. IEEE, 2012, pp. 945–953.

[530] N. Raval, A. Srivastava, A. Razeen, K. Lebeck, A. Machanavajjhala, and L. P. Cox, "What you mark is what apps see," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2016, pp. 249–261.

[531] W. Cui, Y. Kim, and T. S. Rosing, "Cross-platform machine learning characterization for task allocation in iot ecosystems," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2017, pp. 1–7.

[532] D. Xu, Y. Li, X. Chen, J. Li, P. Hui, S. Chen, and J. Crowcroft, "A survey of opportunistic offloading," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2198–2236, 2018.

[533] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, 2016, pp. 201–210.

[534] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network." *IACR Cryptology ePrint Archive*, vol. 2017, p. 35, 2017.

[535] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[536] S. M. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Naval research logistics quarterly*, vol. 1, no. 1, pp. 61–68, 1954.

[537] W. Lee, S. Kim, Y.-T. Lee, H.-W. Lee, and M. Choi, "Deep neural networks for wild fire detection with unmanned aerial vehicle," in *2017 IEEE international conference on consumer electronics (ICCE)*. IEEE, 2017, pp. 252–253.

[538] A. Thomas, Y. Guo, Y. Kim, B. Aksanli, A. Kumar, and T. S. Rosing, "Pushing down machine learning inference to the edge in heterogeneous internet of things applications," 2018.

[539] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[540] A. Mathur, T. Zhang, S. Bhattacharya, P. Velickovic, L. Joffe, N. D. Lane, F. Kawsar, and P. Lió, "Using deep data augmentation training to address software and hardware heterogeneities in wearable and smartphone sensing devices," in *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2018, pp. 200–211.

[541] A. Das, N. Borisov, and M. Caesar, "Fingerprinting smart devices through embedded acoustic components," *arXiv preprint arXiv:1403.3366*, 2014.

[542] A. Mathur, A. Isopoussu, F. Kawsar, R. Smith, N. D. Lane, and N. Berthouze, "On robustness of cloud speech apis: An early characterization," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*. ACM, 2018, pp. 1409–1413.

[543] Z. Chen and B. Liu, "Lifelong machine learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 10, no. 3, pp. 1–145, 2016.

[544] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial intelligence review*, vol. 18, no. 2, pp. 77–95, 2002.

[545] A. Soller, J. Wiebe, and A. Lesgold, "A machine learning approach to assessing knowledge sharing during collaborative learning activities," in *Proceedings of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community*. International Society of the Learning Sciences, 2002, pp. 128–137.

[546] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, and W. Wei, "Ldp-fed: Federated learning with local differential privacy," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 2020, pp. 61–66.

[547] Y. Zhao, J. Zhao, M. Yang, T. Wang, N. Wang, L. Lyu, D. Niyato, and K.-Y. Lam, "Local differential privacy based federated learning for internet of things," *IEEE Internet of Things Journal*, 2020.

[548] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "Hybridalpha: An efficient approach for privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 13–23.

[549] J. Zhang, Y. Zhao, J. Wu, and B. Chen, "Lvpda: a lightweight and verifiable privacy-preserving data aggregation scheme for edge-enabled iot," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4016–4027, 2020.

[550] A. Abdallah and X. S. Shen, "A lightweight lattice-based homomorphic privacy-preserving data aggregation scheme for smart grid," *IEEE Transactions on Smart Grid*, vol. 9, no. 1, pp. 396–405, 2016.

[551] Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6g wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, 2019.

[552] G. Liu, Y. Huang, N. Li, J. Dong, J. Jin, Q. Wang, and N. Li, "Vision, requirements and network architecture of 6g mobile network beyond 2030," *China Communications*, vol. 17, no. 9, pp. 92–104, 2020.

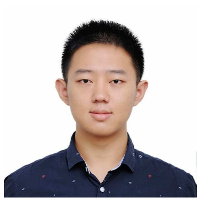
[553] X. You, C.-X. Wang, J. Huang, X. Gao, Z. Zhang, M. Wang, Y. Huang, C. Zhang, Y. Jiang, J. Wang *et al.*, "Towards 6g wireless communication networks: Vision, enabling technologies, and new paradigm shifts," *Science China Information Sciences*, vol. 64, no. 1, pp. 1–74, 2021.

[554] W. Saad, M. Bennis, and M. Chen, "A vision of 6g wireless systems: Applications, trends, technologies, and open research problems," *IEEE network*, vol. 34, no. 3, pp. 134–142, 2019.

[555] K. David and H. Berndt, "6g vision and requirements: Is there any need for beyond 5g?" *IEEE Vehicular Technology Magazine*, vol. 13, no. 3, pp. 72–80, 2018.

Dianlei Xu received the B.S. degree from Anhui University, Hefei, China, and is currently a joint doctoral student in the Department of Computer Science, Helsinki, Finland and Beijing National Research Center for Information Science and Technology (BNRist), Department of Electronic Engineering, Tsinghua University, Beijing, China.

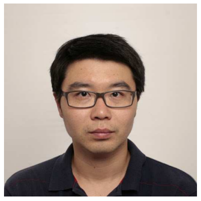
His research interests include edge/fog computing and edge intelligence.



Tong Li received the B.S. degree and M.S. degree in communication engineering from Hunan University, China, in 2014 and 2017, respectively, and the Ph.D. in computer science and engineering from the Hong Kong University of Science and Technology in 2021. He is currently a PostDoc researcher at the Department of Electronic Engineering, Tsinghua University. His research interests include edge networks, ubiquitous computing, and data science. He is an IEEE member.

Yong Li (M'09-SM'16) received the B.S. degree in electronics and information engineering from Huazhong University of Science and Technology, Wuhan, China, in 2007 and the Ph.D. degree in electronic engineering from Tsinghua University, Beijing, China, in 2012. He is currently a Faculty Member of the Department of Electronic Engineering, Tsinghua University.

Dr. Li has served as General Chair, TPC Chair, SPC/TPC Member for several international workshops and conferences, and he is on the editorial board of two IEEE journals. His papers have total citations more than 6900. Among them, ten are ESI Highly Cited Papers in Computer Science, and four receive conference Best Paper (run-up) Awards. He received IEEE 2016 ComSoc Asia-Pacific Outstanding Young Researchers, Young Talent Program of China Association for Science and Technology, and the National Youth Talent Support Program.

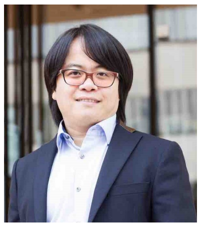


Xiang Su received his Ph.D. in technology from the University of Oulu in 2016. He is currently an associate professor at the Department of Computer Science, Norwegian University of Science and Technology and University of Oulu. He was an Academy of Finland postdoc fellow and a senior postdoctoral researcher in computer science at the University of Helsinki, Finland. Dr. Su has extensive expertise on Internet of Things, edge computing, mobile augmented reality, knowledge representations, and context modeling and reasoning. He is a member of the IEEE.

Sasu Tarkoma received the MSc and PhD degrees in computer science from the Department of Computer Science, University of Helsinki. He is a full professor at the Department of Computer Science, University of Helsinki, and the deputy head of the department. He has managed and participated in national and international research projects at the University of Helsinki, Aalto University, and the Helsinki Institute for Information Technology. His research interests include mobile computing, Internet technologies, and middleware. He is a senior member of the IEEE.

Tao Jiang (M'06-SM'10-F'19) received the Ph.D. degree in information and communication engineering from the Huazhong University of Science and Technology, Wuhan, China, in April 2004. From August 2004 to December 2007, he worked in some universities, such as Brunel University and University of Michigan-Dearborn, respectively. He is currently a Distinguished Professor with the Wuhan National Laboratory for Optoelectronics and School of Electronics Information and Communications, Huazhong University of Science and Technology. He has authored or coauthored more than 300 technical articles in major journals and conferences and nine books/chapters in the areas of communications and networks. He served or is serving as symposium technical program committee membership of some major IEEE conferences, including INFOCOM, GLOBECOM, and ICC. He was invited to serve as a TPC Symposium Chair for the IEEE GLOBECOM 2013, the IEEEE WCNC 2013, and ICCC 2013. He is served or serving as an Associate Editor of some technical journals in communications, including in the IEEE NETWORK, the IEEE TRANSACTIONS ON SIGNAL PROCESSING, the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and the IEEE INTERNET OF THINGS JOURNAL. He is the Associate Editor-in-Chief of China Communications.

Jon Crowcroft (SM'95-F'04) graduated in physics from Trinity College, Cambridge University, United Kingdom, in 1979, and received the MSc degree in computing in 1981 and the PhD degree in 1993 from University College London (UCL), United Kingdom. He is currently the Marconi Professor of Communications Systems in the Computer Lab at the University of Cambridge, United Kingdom. Professor Crowcroft is a fellow of the United Kingdom Royal Academy of Engineering, a fellow of the ACM, and a fellow of IET. He was a recipient of the ACM Sigcomm Award in 2009.



Pan Hui (SM'14-F'18) received his PhD from the Computer Laboratory at University of Cambridge, and both his Bachelor and MPhil degrees from the University of Hong Kong.

He is the Nokia Chair Professor in Data Science and Professor of Computer Science at the University of Helsinki. He is also the director of the HKUST-DT System and Media Lab at the Hong Kong University of Science and Technology. He was an adjunct Professor of social computing and networking at Aalto University from 2012 to 2017. He was a senior research scientist and then a Distinguished Scientist for Telekom Innovation Laboratories (T-labs) Germany from 2008 to 2015. His industrial profile also includes his research at Intel Research Cambridge and Thomson Research Paris from 2004 to 2006. His research has been generously sponsored by Nokia, Deutsche Telekom, Microsoft Research, and China Mobile. He has published more than 300 research papers and with over 17,500 citations. He has 30 granted and filed European and US patents in the areas of augmented reality, data science, and mobile computing.

He has founded and chaired several IEEE/ACM conferences/workshops, and has served as track chair, senior program committee member, organising committee member, and program committee member of numerous top conferences including ACM WWW, ACM SIGCOMM, ACM Mobisys, ACM MobiCom, ACM CoNext, IEEE Infocom, IEEE ICNP, IEEE ICDCS, IJCAI, AAAI, and ICWSM. He is an associate editor for the leading journals IEEE Transactions on Mobile Computing and IEEE Transactions on Cloud Computing. He is an IEEE Fellow, an ACM Distinguished Scientist, and a member of Academia Europaea.