AI Workload Allocation Methods for Edge-Cloud Computing: A Review

Sarah Ammar Rafea [*], Ammar Dawood Jasim [**]

[*] College of Information Eng., Al-Nahrain University, Baghdad, Iraq
Email: sara.ammar@coie-nahrain.edu.iq
https://orcid.org/0009-0000-4220-7374

[**] College of Information Eng., Al-Nahrain University, Baghdad, Iraq
Email: ammar.alaythawy@nahrainuniv.edu.iq
https://orcid.org/0000-0001-5457-8973

Abstract

Edge computing is used with cloud computing as an extension to increase the performance of delay-sensitive applications such as autonomous vehicles, healthcare systems, video surveillance systems, ..etc. The fast increase in the Internet of Things (IoT) devices increases the amount of data transferred in the network. IoT devices are resource-constrained in terms of energy consumption and computation capability. Data processing near IoT devices enabled by edge devices. Hence reduces the transmission power of sending data to the cloud and causes delays due to the cloud being placed far away from the devices. Most real-time applications depend on artificial intelligence (AI) techniques, increasing the computations on IoT-edge devices. Conversely, if this AI workload is executed on the cloud, the delay increase causes degradation in application performance. How to decide where the computation is done in an IoT, edge and cloud network is an important issue. The purpose of optimizing the workload allocation decision is to increase the application performance in terms of Quality of Experience (QoE) and Quality of Service (QoS); hence, the major goal is to reduce the delay time while maintaining the accuracy of the AI systems. As presented in this review, many researchers focus on proposing a workload allocation decision based on AI techniques. In contrast, other research focuses on the AI workload, hence presenting a method for partitioning the AI model to increase the system's accuracy in the resource constraint devices (end device and edge server). Many other researches also used the AI model for resource allocation and provisioning between edge servers and the cloud. In this review, the integration between AI and edge–cloud environment is investigated, the AI workload allocation methods are presented and analyzed, a brief overview of the application of deep learning in edge-cloud computing is also presented, and many challenges that need to be addressed for the AI application are discussed. Many issues and challenges are also presented for optimizing the edge.

Keywords- Cloud computing, Edge computing, Internet of Things, Artificial intelligence, Workload allocation.

I. INTRODUCTION

Internet of things (IoT) devices such as smartphones, sensors, and wearable devices are limited in many aspects, such as energy and computational resources. Offload processing and storage are affected by these limitations. Hence the processes moved from these resource-constrained devices to the cloud since the cloud offers on-demand and scalable service. From other aspects, the cloud is located far away from the devices causing high communication latency [1]. The increasing in IoT applications that depend on real-time computing power and low latency needed edge-computing systems. These applications include healthcare, smart cities, artificial intelligence, robotics, and augmented reality. Edge computing (EC) is a technology that enables both storage and computation near devices. This technology minimizes the latency requirements for the application. For example, for safety-critical vehicular applications, blind intersection warnings, emergency vehicular assistance, lane changing assistance, and forward collision warnings, the maximum acceptable latency is 3 to 10 milliseconds for good performance [2], [3]. Most of the data in EC are processed at the edge, and only a small amount of data is sent to the cloud, causing a reduction in bandwidth requirements. The number of data generated by edge-based IoT applications is huge and continues to grow, which requires intelligent management of this big data [4].
EC facilitates the fluidity of devices and applications spread across various locations. Among its applications are instant analysis of data gathered by portable devices, seamless content dissemination to moving vehicles, and the supervision of the environment using widespread wireless sensor networks (WSN) [1].
Fig. 1 presents the general architecture of the EC system. Edge device which interacts with a human being or the environment to collect data. A good computing capacity is needed for the devices to process the workload. The edge server will be in the same location as the edge device participating in the workload process. An edge gateway also performs network-related functions like protocol translations, tunnelling, firewall protection, or wireless connection to edge devices. The Cloud server is a centralized server

[5]. It is not required for the server to be in the same location or closer to the edge device. Just the critical data will be sent to a centralized server also used for training in case of application requires a complex DL model.

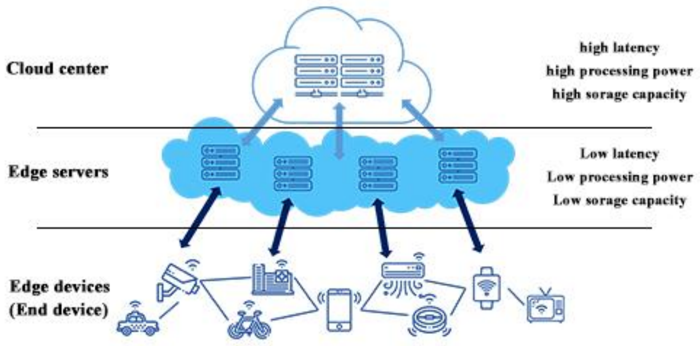

Figure 1: Edge computing system architecture.

The data generated from IoT sensors need to be analyzed in real-time. Artificial intelligence (AI) technology can be used for that purpose. Manly Deep-learning, part of AI technology, requires computation resources to run quickly, so it is implemented on EC to meet the low-latency and high computation requirements of DL on edge devices. It also provides more benefits like bandwidth efficiency, scalability, and privacy. The type of analysis performed on data produced by sensors in an IoT system depends on the particular IoT domain; DL is effective in a number of those domains. A few examples are predicting electricity demand on a smart grid, tracking human activity using wearable sensors, and pedestrian traffic in smart cities. Several IoT applications, like those already stated, provide various streams of data that must be processed and combined, usually including time and spatial correlations that should be employed by machine learning [6]. The workload allocation among edge servers for each Internet of Things (IoT) application influences how quickly requests are processed due to the restricted processing or storage capabilities on the edge devices side. As a result, when the edge server's access devices are deployed widely, the workload distribution becomes a crucial aspect influencing the quality of the user experience (QoE) [7]. The AI application increased and integrated with EC systems in more than one way to increase the application's performance. Many approaches have been adopted to increase the performance of the AI application hence decreasing the response time, increasing the quality of experience, and increasing the application accuracy while decreasing the energy consumption enabling these applications to work in a resource constraint environment.
In this paper, the methods for workload allocation decisions are investigated to present the recent work that focuses on increasing the QoS and QoE of AI applications in an IoT-edge-cloud environment. The methods presented aim to decrease the response time of the application tasks, decrease the energy consumption of end devices, or increase the application accuracy deployed on IoT-edge devices, more details are presented in section VI.
The remainder of this paper is structured as follows: Section II offers an elaborate presentation of EC concepts. Section III presents the AI technology, while Section IV presents the type of workload for AI applications. In Sections V and VI, the inference methods for AI application and the methods for workload allocation are introduced, respectively. The open issues and challenges are presented in Section VII. Finally, the conclusion is presented in Section VIII.

II. EDGE COMPUTING

Lately, there has been a growing fascination with the utilization of edge-clouds for various functionalities. Edge computing (EC), which involves data processing at the periphery of the network, has gained prominence alongside the rapid expansion of the Internet of Things (IoT) and the increasing demand for sophisticated cloud services. EC serves to address the need for prompt responses, constraints on battery life, reduced bandwidth costs, and the assurance of data security and privacy [8]. As data generation at the network's edge becomes more frequent, conducting data processing at this location proves to be more efficient. As cloud computing might not always suffice for data processing in such scenarios, prior initiatives like micro data centers [9], [10], cloudlets [11], and fog computing [12] have been introduced to the community. The term "edge computing" encompasses the technologies that facilitate computation on downstream data for cloud services and upstream data for IoT services, situated at the periphery of the network. In this context, the "edge" pertains to any computing and networking resources positioned between cloud data centers and data sources. For example, smart home gateways serve as intermediaries between inanimate devices and the cloud, while cloudlets, also known as small-scale data centers, function as intermediaries between mobile devices and the cloud. EC emphasizes the need for processing to occur in close proximity to data sources.
EC and fog computing are interchangeable, but EC has a stronger focus on the object's side, while fog computing has a stronger focus on the infrastructure side. Under the EC paradigm, objects fulfill the roles of both data producers and consumers. Things can conduct computing operations from the cloud at the edge and request services and content from it. Edge can distribute requests and delivery

services from the cloud to users and conduct computing offloading, data storage, caching, and processing. To effectively fulfill the requirements for service, such as dependability, security, and privacy protection, the edge itself must be carefully built.

With EC, the aim is to locate the processing close to the data sources. Compared to the conventional cloud-based computing paradigm, this has many advantages. Here, the possible advantages are presented using a few early community results. By shifting processing from the cloud to the edge, researchers created a framework for running a facial recognition application as a proof-of-concept. In [13], The response time was decreased from 900 to 169 ms. Ha et al. [14] found that using cloudlets to offload computational workloads improved response times by 80 and 200 ms for wearable cognitive aid. Moreover, cloudlet unloading could lower energy use by 30% to 40%. The prototype developed by clonecloud in [15] might cut running time and energy consumption for tested apps by 20 percent by combining migration with merging, partitioning, and on-demand instantiation of partitioning across the mobile device and the cloud.

EC systems connect computing, network resources, and storage at the edge of the network, facilitating developers' rapid creation and deployment of edge apps. These systems are motivated by the visions of the IoT and 5G communications. Both industry and academics are paying close attention to EC systems. To investigate fresh research possibilities and aid users in choosing appropriate EC platforms for certain applications. Existing EC systems are divided into three types: push from the cloud, pull from IoT, and hybrid cloud edge, which collectively produce advancements in system design, programming methods, and various applications [16].

In the Push from the cloud category, cloud providers shift computation and services closer to the customer to take advantage of proximity, speed up reaction times, and enhance user experience. Systems like Cloudlet [11], Cachier [17], AirBox [18], and CloudPath [19] are examples. Several conventional cloud computing service providers are actively working to bring cloud services closer to customers and shorten the distance between clients and the cloud to prevent losing market share to mobile EC. While the pull from IoT handles the massive volume of data created by IoT devices, IoT apps pull computation and services from the remote cloud to the network's edge. PCloud [20], ParaDrop [21], FocusStack [22], and SpanEdge [23] are examples of representative systems. Because of advancements in embedded Systems-on-a-Chip (SoCs), many IoT devices are becoming increasingly powerful, allowing them to execute complex algorithms and embedded operating systems. Several IoT device makers incorporate machine learning and even DL capabilities. IoT devices may effectively share processing, network resources, and storage while utilizing EC platforms and tools while maintaining independence. Finally, the Hybrid category represents the combination of cloud and edge benefits. It provides a strategy for achieving worldwide optimal outcomes and minimal response time in modern, sophisticated applications and services. Firework [24] and Cloud-Sea Computing Systems [25] represent representative systems. Cloud services offer the strength and flexibility needed to run complex analytics on IoT data. In contrast, EC solutions employ the processing power of IoT devices to filter, pre-process, and gather IoT data.

III. ARTIFICIAL INTELLIGENCE TECHNIQUES

Artificial intelligence (AI) is a technology that gives machines a certain level of intellect so that they may carry out jobs much like people [26]. Even though data mining (DM) and heuristic-based algorithms [27] have been major contributors to AI for IoT solutions throughout the years, it is important to note that while DM and ML both use huge amounts of data, ML and DL focus on simulating the human learning process. In contrast, DM is intended to extract the rules from the data [28], [29]. DL is a higher-level intelligence and indicates the future direction of AI compared to DM.

DL is a subfield of machine learning that focuses on developing AI systems using programming techniques modelled after the human brain's anatomy and physiology. DL can handle vast amounts of data and allows computers to make judgments in the same way humans do, hence efficiently performing tasks such as prediction and classification [30]. Because of its multilayer nature, DL has a more potent capacity to extract high-level characteristics from large amounts of data [31]. Various DL applications and goals have led to the development of several DL architectures. Architectures within the domain of DL, such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks, convolutional neural networks, and transformers, have found extensive applications across diverse fields. These applications encompass board game algorithms, computer vision tasks, the realms of the Internet of Things (IoT), speech recognition systems, natural language processing, robotics, autonomous vehicles, healthcare applications, and the domain of machine translation [32]–[35]. They are also used in various industries, including finance, retail, and marketing, to solve complex problems and improve decision-making.

In edge-cloud computing, DL algorithms are typically deployed on edge devices or gateways, able to transfer data and models between the edge and cloud as needed enabling real-time processing and decision-making while reducing the latency and bandwidth requirements of transmitting data to the cloud. Neural networks in DL are designed with multiple layers, allowing the network to learn hierarchical representations of data. This allows DL algorithms to perform highly accurate tasks such as image and speech recognition, natural language processing, and predictive modelling. The trained of DL algorithms can be either supervised or unsupervised learning tasks. Supervised learning algorithms are trained on labeled data, where the desired output is already known. On the other hand, unsupervised learning algorithms are trained on unlabeled data and attempt to find patterns and relationships in the data without prior knowledge of the output. The success of DL is largely due to the availability of large amounts of labeled data, as well as advances in computing power and algorithms that make it possible to train deep neural networks in a reasonable amount of time. However, DL algorithms can also be computationally intensive, requiring specialized hardware and software to run effectively.

The emergence of numerous DL-based intelligent services and applications has significantly transformed various facets of human life, primarily owing to the inherent advantages of DL in natural language processing (NLP) and computer vision (CV). These breakthroughs are not only a consequence of DL's advancement but also intricately linked to the rapid expansion of data availability and processing capabilities. However, despite these achievements, the accessibility of intelligent services remains limited for several use cases, such as smart city initiatives, the Internet of Vehicles (IoVs), and more, largely due to cost-related factors. Consequently, the training and inference procedures of DL models in the cloud necessitate the transmission of substantial data volumes from devices or users to the cloud, resulting in significant network bandwidth consumption. Moreover, the issue of latency has impeded the widespread deployment of intelligent services. This is particularly evident in time-sensitive applications, like collaborative autonomous driving, which demand swift access to cloud services, a feature that is typically not assured and might not always be available [37]. Reliability is also important because many industrial scenarios require highly reliable intelligent services, even if network connections are lost. Most cloud computing applications rely on wireless communications and backbone networks to connect users to services. Furthermore, privacy is important since the data needed for DL may contain a lot of sensitive information. Smart cities and households depend on privacy protections.

Allocating workloads is one of the most effective and efficient strategies to boost the performance of the cloud computing system. Moreover, it examines one of the most serious challenges of our day [38]. Several methods have been proposed to overcome this assumption, and the most effective ones have been applied by utilizing DL technology [39]. Also, optimizing the edge to implement the intelligent edge is an important issue in recent research, enabling the complex AI model to work efficiently in resource constraint devices. On the other hand, decision-making plays an important role in application performance in the edge cloud system since many recent researches focus on workload allocation decisions among end devices, edge servers, and cloud centers.

IV. TYPE OF AI APPLICATION WORKLOAD

As mentioned previously, the AI application benefits greatly from EC. In addition, DL algorithms can be used in congestion with edge-clouds to enhance management and provide better performance from many applications, especially for delay-sensitive applications. This combination is used by various researchers in various projects, such as autonomous vehicles, medical devices, smart watches, IoT sensors, etc. The major applications that benefit from the edge are presented in this section. This application, in general, generates AI workload by its end devices and is sent to the edge–cloud server for processing. Such as these applications are Computer Vision, including object detection and recognition, image and video compression, natural language processing, Predictive Maintenance, Anomaly Detection, and Personalized Recommendations.

A- Computer Vision

DL algorithms can be used for image classification, object detection, and recognition, which is the fundamental task of computer vision on edge devices such as cameras or mobile devices. In addition, the edge server enables real-time image and video stream processing. For example, video surveillance, item counting, and vehicle recognition require computer vision tasks. Such data comes naturally from cameras at the network edge [40]. The frame rate of real-time inference in computer vision can sometimes be described in terms of the camera's frame rate, typically 30 to 60 frames per second [41]. Also, uploading camera data to the cloud raises privacy issues, particularly if the video frames contain private documents or private information, which is another reason to do computing locally. The third benefit of EC for computer vision jobs is scalability. If many cameras transmit huge video streams, the uplink's bandwidth to a cloud becomes restricted. Vigil [42], an innovative illustration of an edge system for computer vision application that utilizes the power of edge computing. The Vigil system comprises a wireless camera network that captures images and sends them to edge EC nodes for processing. The EC nodes then use intelligent algorithms to select the frames that require further analysis, reducing the amount of data that needs to be transmitted to the cloud. In Vigil, EC is used for two reasons: to scale as the number of cameras grows and to consume less bandwidth than the naive strategy, which is uploading all frames to the cloud for analysis. The edge-based video analysis is also motivated by scalability in VideoEdge [43]. They employ a tiered architecture of edge and cloud compute nodes to aid with load balancing while retaining high prediction accuracy. Commercial devices, like Amazon DeepLens [40], employ an edge-based strategy, performing the detection of images locally to reduce latency time and only uploading scenes of interest to the cloud for distant viewing if an intriguing object is discovered, thereby conserving bandwidth [6].

B- Natural Language Processing

DL algorithms can be used to perform natural language processing tasks, such as sentiment analysis and language translation, on edge devices. DL has also gained popularity for tasks involving natural language processing [44], such as speech synthesis [45], named entity recognition [46] (which involves recognizing the many components of a phrase), and machine translation [46]. Recent systems have been able to attain latency for conversational AI on the order of hundreds of milliseconds [47]. Visual question-and-answer systems [48] exist at the nexus of computer vision and natural language processing, is to ask queries about images and obtain natural language responses. Depending on how information is displayed, different latency requirements apply; For instance, conversational responses are best given within 10 milliseconds, whereas a response to a typed Web inquiry can wait up to 200 milliseconds [49]. As Examples of natural language processing applications performed on edge are Voice assistants like Apple Siri and Amazon Alexa.

Even if some of the processing carried out by these voice assistants takes place in the cloud, wake words like "Hello Siri" are recognized by on-device processing. If the wake-word is found, the speech recording is only forwarded to the cloud for additional processing, query answer, and interpretation. Two on-device DNNs are used in AppleSiri's wake-word processing to categorize voice, including silence, general speech, and wake-word [50]. The first DNN uses a low-power and has fewer layers (5 levels, 32 units). When the output of the first DNN exceeds a certain level, the main processor activates a second, more potent DNN (5 layers, 192 units). Methods for wake-word recognition need to be further modified to function on even less powerful devices, like an Arduino or smartwatch. A single DNN is utilized on the Apple Watch, and its hybrid structure borrows from the two-pass method. Researchers at Microsoft optimized an RNN model for wake-word ("Hello Cortana") detection to fit in a small amount of RAM for voice processing on an Arduino [51]. Overall, while EC is employed for wake-word recognition on edge devices, a professional translator can interpret 5 times faster. The latency also needs to be addressed for more complicated natural language jobs [6].

C- Network Functions

It has been suggested to use DL for network operations, including intrusion detection [52], [53], and wireless scheduling [54]. By definition, these systems operate at the network edge and have strict latency requirements. For instance, to prevent establishing a bottleneck, An intrusion detection system must perform detection at a line rate, such as 40 s, to actively respond to a detected attack by blocking malicious packets [55]. Yet, the intrusion detection system's latency requirements are less stringent if it runs in the passive mode. To instantly decide which packets to send where a wireless scheduler needs to run at a line rate. An additional example of a network function that can make advantage of DL at the network edge is network caching. In an EC situation, various end devices in the same geographic area could make numerous requests to the same remote server for the same material. The perceived response time and network traffic can be greatly decreased by caching such items at an edge server. Deep reinforcement learning or DL for content popularity prediction are the two most common ways to implement DL in a caching system [56]. For instance, Saputra et al. [57] employed DL to forecast the popularity of content. The cloud gathers data on content popularity from all of the edge caches to train the DL model. On the other hand, deep reinforcement learning for caching ignores popularity prediction and relies only on reward signals from its activities. To train deep reinforcement learning for caching, Chen et al. [58] used the cache hit rate as the reward [6].

D- Internet of Things

In many industries, including smart cities, the grid, and wearables for healthcare, an automatic understanding of IoT sensor data is sought. The specific IoT domain will determine the type of analysis done on these data, but DL has proved successful in several of them. As examples, consider pedestrian traffic in a smart city [59], electrical demand prediction in a smart grid [60], and human activity detection from wearable sensors [58]. One distinction in the IoT environment is that several data streams might need to be combined and processed and that machine learning should use the space and time correlation that these data streams often have. One framework for fusing IoT data that takes advantage of spatiotemporal correlations is DeepSens [59]. Convolutional neural networks (CNNs) are used in a hierarchy in the DeepSens architecture to record numerous sensor modalities, while recurrent neural networks (RNNs) are used to collect temporal correlations; this generic architecture may be used for a variety of applications that call for numerous sensor inputs, such as vehicle tracking, the detection of human activity, and biometric identification. Another field of study in the IoT and DL focuses on compressing DL models to fit onto computationally constrained end devices like the Raspberry Pi or Arduino, which often have little memory and low-power CPUs. This line of research addresses the challenge of deploying DL models to devices with constrained resources. Techniques such as pruning, quantization, and knowledge distillation have been proposed to reduce the size of DL models while maintaining acceptable accuracy levels. These techniques have the potential to bring the benefits of DL to resource-constrained devices and enable innovative IoT applications. DeepThings [61] tests with the Raspberry Pi 3, while DeepIoT [62] utilizes Intel's Edison board. In the survey by Mohammadi et al., additional instances of using DL in IoT contexts, such as in industry, agriculture, and smart homes, are provided [63]. When IoT sensors are located in public areas, like the Hudson Yards smart city construction in New York City, serious privacy problems might arise; these concerns can be addressed using EC combined with IoT devices. The technology seeks to combine a variety of sensors, such as those that measure temperature, noise, and air quality, coupled with cameras to provide marketers an estimate of the number of individuals who watched commercials and how long they were seen, as well as their mood based on facial expressions. The potential for misuse of such data raises serious concerns about privacy infringement, highlighting the need for greater security and privacy protection measures. EC can help address these concerns by processing data locally on the device, reducing the need to transmit sensitive information to centralized servers. By performing computation and analysis closer to the source of data, EC can help protect sensitive information, reducing the risk of data breaches and other privacy violations. The adoption of EC in IoT devices can help enable innovative and valuable IoT applications while ensuring user privacy and security. Nonetheless, privacy advocates have issued serious cautions in response to this [64]. IoT processing on the edge is driven by privacy concerns, even if real-time analysis of IoT sensor data is not necessarily required and sensor connection bandwidth needs are often low (unless cameras are involved) [6].

E- Virtual Reality and Augmented Reality

DL has emerged as a promising approach to predict a user's perspective within the immersive world of 360-degree virtual reality (VR) in order to create a seamless and engaging user experience. This involves the real-time computation of predictions, enabling the selection of the most relevant sections from the 360° video content [65], [66]. DL also finds utility in the realm of augmented reality (AR), where it serves to identify objects of interest within the user's field of vision and seamlessly overlay virtual elements onto the real world. In both AR and VR scenarios, the critical performance metric is the "motion-to-photons" delay, representing the time lag between a user's movements and the corresponding update on the display [67], [68]. This delay typically needs to be minimal, ranging from tens to hundreds of milliseconds [69]. The motion-to-photons latency acts as an upper limit for the timing demands imposed on DL because various other components within the AR/VR system can introduce substantial latency [70]. Furthermore, the specific application and nature of user interactions can also influence the motion-to-photons delay requirements. Ensuring satisfactory performance often necessitates edge computing (EC) to prevent significant delays, as offloading AR computations to remote servers could result in delays spanning hundreds of milliseconds [6], [71].

V. AI WORKLOAD INFERENCE PROCES

Several methods have been utilized to speed up the processing of the tasks generated by the AI application, hence speeding up the inference process. To meet the stringent latency requirements of various applications, several architectures have been proposed for performing deep neural network (DNN) inference. These architectures are designed to optimize the computation and memory requirements of DNN models, enabling them to perform inference with minimal latency. There are three types of computation locations. The first type is by executing the task on-device, where DNNs are run on the end device. The second type involves carrying out the workload in edge server-based architectures, where data is executed on edge servers. The last type involves collaborative computation involving end devices, edge servers, and the cloud [6].
Moreover, the speed-up execution of the tasks can be done by enhancing the model design, model compression, or hardware specification in case of execution on the end device. In the case of execution in an edge server, the speed-up process can be achieved by applying preprocess and resource management. Finally, in the case of computing across layers (end device- edge server, cloud), optimizing the offloading, DNN partitioning, distributed computing, and workload allocation decisions among the layers. These topics will be covered in more detail in the next sections.

A- Execution on the edge device

The DL algorithm's delay when applied to a device with limited resources has been the subject of extensive research. By lowering the DNN's latency while running on the end devices or edge servers, such initiatives could positively affect the entire edge ecosystem. Regarding the model design, the researchers focus on producing models with a smaller number of parameters in the deep neural network (DNN) architecture. This is particularly important when building DNN models for resource-constrained devices, where reducing memory usage and execution latency is critical. However, reducing the number of parameters can also reduce model accuracy. Therefore, researchers aim to balance model complexity and accuracy, seeking to produce models optimized for specific devices and applications. By optimizing DNN models for resource-constrained devices, machine learning researchers can enable new applications that were previously impractical, opening up new opportunities for innovation in fields such as IoT, mobile devices, and wearable technology. There are numerous methods for doing this, description of a few well-liked computer vision-based DL models for devices with limited resources is presented. These models include MobileNets [72], solid-state drive (SSD) [73], YOLO [74], and SqueezeNet [75]. The convolution filters are divided into two easier operations by MobileNets, which lowers the number of computations required. SqueezeNet uses specialized convolution filters to downsample the data. Single-shot detectors like YOLO and SSD anticipate the class and location of the object simultaneously, which is substantially faster than doing these tasks one at a time. To facilitate quick bootstrapping, users can leverage pre-trained models with pre-trained weights, readily available from open-source machine learning systems such as Tensorflow [76] and Caffe [77]. While in the case of the Model Compression aspect, considered as another method for enabling DNNs on edge devices is by compressing the DNN model. These techniques often aim to compress the existing DNN models with minimal compromise on accuracy in relation to the original model. Techniques for compressing models, such as parameter quantization, parameter trimming, and knowledge distillation, are commonly employed. The following is a succinct overview of various strategies. By switching from floating-point values to low-bit width numbers and compressing the parameters of an existing DNN, parameter quantization avoids the expensive floating-point multiplications., pruning entails deleting the least significant parameters (such as those that are almost zero). Both solo and cooperative considerations of the quantization and pruning techniques have been made [78]. DeepIoT [79] and CMSISNN [80] are techniques for optimizing deep learning structures for deployment on edge and mobile devices. DeepIoT prunes DL structures, removing unnecessary connections to reduce complexity. CMSISNN quantizes DL models, reducing precision to save memory and improve performance on ARM Cortex-M processors commonly used in IoT devices. Both aim to improve performance and reduce resource requirements for DL models on IoT devices. To hasten DNN performance, it additionally improves data reuse in matrix multiplication. For an RNN model, Han et al. [81] suggested quantization and pruning, resulting in a 10 speedup and quantization inducing a 2 speedup. Lane and Bhattacharya [82] compacted the neural network, and a smaller DNN that mimics the behavior of a larger, more potent DNN is created by knowledge distillation [83]. To accomplish this, the smaller DNN is trained using the output predictions generated by the larger DNN. Fundamentally, the reduced-size DNN closely approximates the function learned by the larger DNN. Another technique known as fast exiting computes the initial layers and leverages the outcomes to produce approximate classification results [84]. The combinations of these model compression strategies have been examined in several papers. To meet application needs and adhere to mobile resource limits, Adadeep [85] automatically selects between various compression algorithms, including pruning and the unique filter structures borrowed from SqueezeNet and MobileNet. Quantization and GPU caching of results from intermediary layers are combined in DeepMon [86]. To reduce duplicate computations and speed up execution, caching makes use of the observation that a visual input's variation across frames is minimal. As a result, it is possible to reuse previously computed results in the current frame. Finally, in the Hardware improvement aspect, To expedite DL inference, hardware makers are utilizing already-existing technology like GPUs and CPUs as well as creating specialized application-specific integrated circuits (ASICs), like Google's Tensor Processing Unit (TPU) [87]. Another recently suggested custom ASIC is ShiDianNao [88], which focuses on energy- and latency-efficient memory accesses. It is part of the DianNao [89] family, a collection of DNN accelerators. Still, it is designed for embedded devices, which is advantageous in the context of EC. Based on a field programmable gate array (FPGA), DNN accelerators are still another promising strategy since FPGA allows for quick processing while keeping flexibility [88]. These specialized ASIC and FPGA designs are typically more energy efficient than conventional GPUs and CPUs, which are built to support a wide range of applications while consuming more energy. For application developers to take use of the hardware accelerations, vendors also offer software tools. To take advantage of Intel processors, which include Intel's GPUs, CPUs, FPGAs, and visual processing unit, chip manufacturers have produced software tools, such as Intel's OpenVINO Toolkit [90], [91]. Another entrant in this market is Nvidia's EGX platform [92], which supports a variety of Nvidia hardware, from compact Jetson Nanos to potent T4 servers. The Neural Processing software development kit (SDK) from Qualcomm is made to work with their Snapdragon processors [93]. There are other universal libraries designed for mobile devices that are independent of any particular hardware, such as RSTensorFlow [94], which accelerates DL matrix multiplication using the GPU. Moreover, software methods for effectively utilizing hardware have been devised. For example, Lane et al. [95] accelerated execution by decomposing DNNs and allocating them to different local processors (such as GPU and CPU). The good survey by Sze et al. [96] provides more information on hardware-accelerated DL.

B- Edge Server Computation

Using big, powerful DNNs with real-time execution requirements on edge devices is still difficult owing to resource constraints (e.g., power, memory, and compute), even with the hardware speedup and compression strategies discussed above. So, it becomes sense to think about shifting DNN calculations from end devices to stronger systems, like the cloud or edge servers. The cloud is ineffective for edge applications that demand quick responses, though [97]. The edge server becomes the go-to assistant because it is close to the users and can react to their requests rapidly. Offloading all computing from end devices to the edge server is the simplest way to use the edge server. When this happens, the end devices send their data to a nearby edge server, where it is processed and then receive the results. To assess wireless signals, Wang et al. [98], for instance, constantly offloaded DNNs to the edge server. Data preprocessing can help reduce data redundancy when transmitting data to an edge server, reducing communication time. Glimpse [99] controls which camera frames are offloaded using change detection while offloading all DNN computation to a nearby edge server. Glimpse will carry out frame tracking locally on the end device if no changes are found. Real-time object identification on mobile devices is made feasible by this preprocessing, which enhances the capability of the system to analyze data. Similarly, Liu et al. [100] propose two preprocessing procedures used to create a food identification system: first, fuzzy pictures were removed, and then the image was cropped only to contain the relevant elements. Both preprocessing phases can reduce the quantity of data offloaded and are rapid. In computer vision, feature extraction is a standard preprocessing step, and because DNNs function as feature extractors, it does not applicable in the context of DL. Also, Edge Resource Management is utilized. Hence DNN tasks from various end devices must run and be effectively managed on shared computing resources when DNN computations are performed on edge servers. The compromises between precision, speed, and other performance indicators, like the number of requests served, have been the focus of numerous studies looking into this problem field. A pioneering endeavor in this domain, Video Storm [101], assesses these trade-offs to determine the optimal DNN configuration for each request, aligning with the objectives of minimizing latency while preserving accuracy. Additionally, Chameleon [102] enables the real-time modification of configurations during the input streaming of videos. Furthermore, VideoEdge [43] explores the joint optimization of all DNN hyper parameters and distributed computation across a hierarchy of cloud and edge servers. Mainstream addresses a similar issue of balancing accuracy and latency trade-offs on edge servers [103]. Still, their solution uses transfer learning to cut down on the CPU power required for each request. Furthermore, transfer learning allows different applications to use the DNN model's lower common layers while computing higher layers that are specialized to each application, lowering the overall computation required.

C- Computing Across Edge Devices

While edge servers can improve the processing speed of DNNs, intelligent offloading can sometimes be used as an alternative to having edge devices run DNNs on the edge servers. There are various offloading scenarios to consider, such as binary offloading, which involves making a decision on whether to offload the entire DNN or not. Another scenario is partial offloading, where partitioned DNNs are considered, and a decision is made on which portion of the DNN calculations should be offloaded. This can be especially useful for edge devices that have limited computational capabilities, as offloading certain parts of the DNN to more powerful devices can help improve inference times while reducing the burden on edge devices. By employing intelligent offloading techniques, developers can achieve optimal performance for their DNN applications while taking into account the limitations and capabilities of their edge devices. The third option is Hierarchical Architectures, where offloading is performed across a combination of edge devices, edge servers, and cloud, and finally, Distributed Computing Approaches, where the DNN computation is distributed across multiple peer devices.

Offloading the DNN model, current methods, like Deep Decision [104], [105], and MCDNN [106], utilize an offloading technique based on optimization with restrictions on network latency and bandwidth, device energy, and cost. These choices are based on evaluating the trade-offs between the various DNN models' input size, accuracy, latency, and energy consumption. The selection of various DNN models can be made from the existing, well-liked models, or new model variants can be created by distilling existing information or by "mixing and matching" DNN layers from various models [106]. Even in the context of EC [107], Offloading is a well-researched topic in networking, and it has also been explored in the context of deep neural network (DNN) offloading. One of the benefits of DNN offloading is the added flexibility of choosing not just where to run the model but also which specific DNN model or part of the model to execute. However, determining whether to offload or not depends on various factors, such as the amount of data, available technology, the specific DNN model being used, and the quality of the network. Making the decision to offload requires careful consideration of these factors to ensure that the chosen approach will be efficient and effective. As technology evolves and new DNN models are developed, researchers will continue to explore the potential benefits and limitations of offloading and develop new techniques to optimize the process for different applications.

Partitioning the DNN Models, A Fractional Offloading technique that takes advantage of DNNs' distinctive structure, particularly their layers, can also be considered. These model partitioning techniques involve computing certain layers on the device and others by an edge server or in the cloud. Utilizing the computing cycles of other edge devices may potentially reduce latency; however, it is important to ensure that the delay in relaying intermediate findings to the DNN division point still results in overall net gains. The idea behind model partitioning is that once the DNN model's initial few layers have been calculated, Since the intermediate findings are very small in size, sending them to an edge server over the network is quicker than sending the initial raw data [61]. The methods that divide after the initial layers are motivated by this. One study that intelligently chooses how to divide the DNN into layers while considering network conditions is Neurosurgeon [108]. The DNN can be divided into segments along the input dimension in addition to layers. Because the input and output data sizes and the memory footprint of each partition may be freely set, rather than the minimum partition size being determined by the discrete DNN layer sizes, such input-wise partitioning enables fine-grained partitioning. This is crucial for ultra-lightweight gadgets like IoT sensors, which might not have enough memory to store a full DNN layer. Nevertheless, input-wise partitioning might lead to greater data reliance because computing subsequent DNN layers require data from nearby partitions. MoDNN [109] and DeepThings [61] are two examples of input-wise partitioning. These partial offloading methods using DNN partitioning are generally similar to earlier, non-DNN offloading methods like MAUI [10] and Odessa [110], which break down an application into its jobs and choose which tasks to run where depending on energy and/or latency concerns. The choice of how to divide the constituent subtasks in the DL scenario is new, as the DNN can be separated into layers, inputs, or perhaps other yet-to-be-explored ways.

In the case of Edge Devices and the Cloud, DL computation can be carried out both on edge devices and in the cloud. While purely offloading to the cloud may not meet the real-time demands of the DL applications under consideration, careful use of the cloud's potent compute capabilities may be able to reduce the overall processing time. Approaches in this domain frequently take into account DNN partitioning, where certain layers can function on an edge server, an end device, or the cloud, as opposed to making a binary decision on whether to perform computation on the edge server or cloud. The DNN model was split in two by Li et al. [31]; the edge server computes the lower layers of the DNN model, and the cloud computes the higher layers. After receiving and processing the input data using lower-layer DNN, the edge server delivers the intermediate results to the cloud. After computing the upper layers, the final results are sent back to the end devices via the cloud. Such solutions use both the edge server and the cloud, with the cloud helping with computationally demanding queries and boosting the edge server's processing pace while reducing network traffic between the cloud and the edge server. Moreover, DDNN [111] combines this with the quick departing notion to spread processing across a hierarchy of end devices, edge servers, and the cloud. So that not all calculation requests are sent to the cloud.

The edge server frequently serves clients located in a small geographic region, indicating that their input data and, thus, their DNN outputs may be comparable, which is a distinctive feature of EC. In the case of image recognition, Precog [112] leverages this comprehension by deploying more concise, targeted image classification models onto end devices, guided by recent insights from other devices serviced by the same edge server. Should the on-device classification prove unsuccessful, the query is relayed to the edge server with all classification models available. They deliberate on the potential application of their classification model placement decisions to DNNs, despite their evaluation not directly involving DNNs. This approach entails a fusion of both weaker and stronger classification models, akin to knowledge distillation for compressed models, providing a comprehensive examination of the specific models required on end devices within edge environments.

The methodologies outlined above primarily take into account offloading processing from end devices to other, more capable devices (e.g., the cloud and the edge servers). Another approach to addressing the challenge of DNN processing in resource-constrained environments involves distributed computing. In this case, DNN computations can be distributed across multiple edge devices to improve processing time and reduce the workload of individual devices. For instance, DNN executions are distributed utilizing fine-grained partitioning on lightweight end devices like Android smartphones and Raspberry Pi by MoDNN [109] and DeepThings [61]. Based on the end devices' computational power and/or memory, the DNN partition decision is made. With DeepThings creating a load-balancing heuristic and MoDNN employing a methodology similar to MapReduce, the input data are distributed to helpers at runtime under load-balancing principles. Online adjustments can be made to the data assignment to the assistance devices to consider dynamic shifts in the availability of computational resources or network conditions. More formal distributed system approaches could be used to provide verifiable performance guarantees in these cases.

VI. Methods for Workload Allocation

Cloud computing was the first technology to offer computation offloading. The definition states that the execution of some or all computing duties is delegated to the cloud from terminal devices with low processing capability. The issue of terminal devices with low processing capability delegating some or all of their computing activities to the edge is known as computation offloading in EC [113]. The primary factors to be taken into account are whether or not which nodes, how much, and to which terminal devices will offload are all defined. Offloading computing tackles the problems of few resources and excessive energy consumption in terminal devices. The assumption that the default server has adequate processing capacity and is unconcerned with its energy usage or network stability is one of several made by traditional techniques of offloading computation in cloud computing. The computational offloading issue in EC, where edge devices and servers have constrained processing capabilities, cannot be solved using standard approaches based on the above assumptions [114]. Offloading techniques for computation that are reasonable can cut down on latency and energy use. As a result, computation offloading is a crucial study area for EC optimization [115].
This section presents the recent work on workload allocation decision methods. Table 1 summarizes the methods considered in this work for workload allocation. The workload allocation decisions depend on many factors, such as network condition, device capability, model complexity,...etc. Where to execute a given workload is considered an np-hard problem, especially in the multi-end device and multi-server case. Many research efforts have been made to reduce energy consumption, decrease delay response time, improve result accuracy, or achieve more than one of these goals. Many researchers use a mathematically based approach for workload allocation decisions. In [116], Investigations are made on the workload distribution issue in an IoT edge-cloud computing system. They develop a latency-based task allocation issue that offers the best workload distributions across local edge servers, neighboring edge servers, and the cloud to ensure low energy usage and delay. Then, a delay-based workload allocation (DBWA) algorithm based on Lyapunov drift-plus-penalty theory is used to solve the issue. Also, To determine the best decision, several other researchers use the Lyapunov optimization technique [117]–[119]. Resource allocation and compute offloading are also seen in certain research as optimization issues, such as linear programming [120] and mixed integer non-linear programming [121]–[123]. Other traditional methods include the alternating direction method of multipliers (ADMM) [124], Stack-elberg game [125], and so on. These methods are mathematically complex and computationally intensive compared to the methods that use AI technology for offloading decisions as present in Fig. 2.

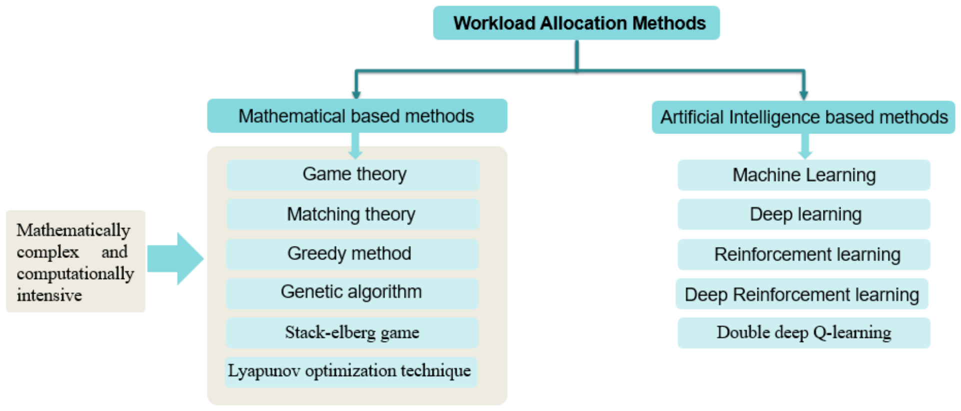
Workload Allocation Methods
Mathematical based methods
Artificial Intelligence based methods
Mathematically complex and computationally intensive
Game theory
Matching theory
Greedy method
Genetic algorithm
Stack-elberg game
Lyapunov optimization technique
Machine Learning
Deep learning
Reinforcement learning
Deep Reinforcement learning
Double deep Q-learning

Figure 2: Workload allocation methods.

In [126] and [127], the authors present two algorithms to optimize the workload allocation for AI applications. The first algorithm is for a single workload, and the second is for a multi-job workload. The algorithms aim to reduce the response time for latency-sensitive applications by deciding where to deploy the online process among the device, edge server, or the cloud. The experiment used six edge AI workloads from Edge AIBench experiments [128]. The AI workloads are short-of-breath alerts, patient phenotype classification, life-death prediction, face recognition, lane detection, and action detection. The author also used priority as a factor for making a decision. Hence in a multi-job scenario, each workload has a different priority. The type of AI model and dataset differs for each application chosen based on the application. The training process is always implemented on the cloud (offline process) then the pre-trained model sends to the devices. The online process includes inference and prediction. The algorithm depends on four factors to determine where to execute the inference process: the computational ability of the cloud server, edge server, and device, the complexity of the AI model, the size of the dataset, and the network latency. The proposed algorithm is compared with the three-based method of execution (execute all workload on the cloud, execute all the workload on the edge device or on the device). The result shows that the multi-job algorithm outperforms the other approach by 33% up to 63%. In [129], the authors proposed a method consisting of two-stage to allocate the workload to the edge server or the cloud server. The edge server, called the auxiliary network, contains a lightweight DNN, while the cloud server, called the principle network, contains a powerful DNN. The authors present a method to estimate the confidence level (trustworthiness) of the prediction made in the auxiliary network. If the confidence level of

the prediction is higher than the predefined threshold, then there is no need to send the workload to the principal network. If the confidence level is lower than the predefined threshold, then the workload should send to the principal network for more accurate prediction. The threshold is determined based on kernel density estimation (KDE). The authors balance the accuracy with the power constraint of the edge devices; hence if more workload is processed in the auxiliary network, the power consumption is reduced since there is no transmission to the principal network, but at the same time, the accuracy of predictions is reduced since the DNN model is lightweight. The powerful DNN model is computationally heavy compared to the power constraint of the edge. In the case of executing all workloads in the principal network, power consumption increased because of the principal network's transmission but achieved high inference accuracy. The DNN models used for the experiment are residual network (ResNet) [128] and wide residual network (WRN) [131], with the data sets CIFAR-10 and CIFAR-100 [132]. The proposed methods are compared with two methods which are CDL [133] and BranchyNet [132]. The authors used two combinations of a network. The first one is ResNet-18 as an auxiliary network and ResNet-110 as a principal network. In the second combination, WRN-28 as an auxiliary network, and WRN-10 as a principal network. The author also used the big Snapdragon 835 core to estimate the power consumption of the edge as well as the Wi-Fi model as the transmission media model. Utilizing the entire energy utilized by the application processor, the transmission medium, and the overall latency of inference with the transmission, the power consumption of the edge device is assessed. In comparison to earlier studies, the results demonstrate an accuracy improvement of up to 3.93%. In [134], the authors present a machine learning-based approach to workload allocation decisions in vehicular EC, the authors used a two-stage mechanism based on the classification of the task will be success or failure if it's executed on edge or cloud respectively. While the second stage is predicting the response time for the execution in each layer and selecting the minimum response time layer. For the prediction layer, a linear regression model was used. The metric that the author depends on is the network bandwidth, data size, and delay time. In [135], the authors proposed an algorithm to reduce energy consumption and task processing time for IoT scenarios in mobile EC environments, the authors suggest a dynamic cost model. the suggested algorithm TEMS, which takes into account energy use, processing time, data transmission costs, and energy in standby devices. To get better execution times at cheaper costs, the task scheduling selects a cloud or mobile edge computing (MEC) server or local IoT device. The evaluation of the simulated environment reduced energy usage by up to 51.6% and increased task completion efficiency by up to 86.6%. In [136], the authors suggest an IoT-edge-cloud network computation offloading strategy using mobile vehicles. The vehicles determine whether to calculate the tasks locally, on a MEC server, or in a cloud computing center once the sensing devices generate and transmit the tasks to them. Decisions on compute offloading are made using the deep reinforcement learning approach based on the utility function of energy usage and transmission latency. The experimental findings demonstrate that the suggested strategy maximizes reward and minimizes delay while implementing task offloading of sensing devices by fully using the existing infrastructures. In [138], the binary offloading decision between the wireless device and edge server is presented to minimize the MEC network's system utility. Use the deep supervised learning-based computational offloading (DSLO) technique for dynamic computational workloads in MEC networks. To faster the model convergence process and increase the model's robustness, batch normalization is also used. The authors take into account a MEC network with several wireless devices and a single edge server, where each MEC scenario is distinguished by a certain job weighting factor. Each wireless device in the network decides in real-time whether to conduct computing activities locally or offload them to the edge server. The numerical findings demonstrate that DSLO can swiftly adapt to new MEC conditions and only needs a small number of training samples. In [138], the authors propose a two-stage workload allocation method for an IoT-edge-cloud environment. The proposed methods, Distributed Deep Meta learning-driven Task Offloading (DDMTO), are based on deep neural network (DNN) for two-stage binary decision and deep meta-learning for training the initial parameter model. The model decides if to compute the task locally or offload the task to the server. If the decision is to offload the task, another decision will make to decide if to execute on the edge server or the cloud. The result shows that the DDMTO method outperforms the other earlier strategy, such as MO-BFO, DMRO, and DDTO. In [139], The authors proposed a method based on Double Deep Q-Network (DDQN) algorithm for workload allocation in an edge-cloud environment called DDQNEC. The decision makes to send the tasks to the edge server or the cloud center. The objective is to minimize delay and meet computation and communication needs in edge-cloud computing by employee one of the reinforcement learning method. DDQNEC makes offloading decisions dynamically depending on resource utilization, network state, and task constraints. Simulation results demonstrate that DDQNEC outperforms heuristic approaches regarding task failure, task offloading, and resource utilization. In [140], The author provides a workload allocation method to tradeoff between energy consumption and delay. The proposed method called deep Post-Decision State (PDS)-learning, that depend on Deep Q-Network (DQN) for IoT-edge-cloud computing. The proposed method outperforms multiple other methods considered in the work in terms of job failure rate by 5.7%, delay by 4.5%, cost by 4.6%, energy consumption by 3.9% and computational overhead by 6.1%.
The recommendation to use a specific method rather than others depends on the used scenario and application. Generally in a very high dynamic environment with a high number of edge devices and end devices the reinforcement learning, Q-learning and deep learning models show high performance compared to the baseline approach while in a static environment a linear regression model show enhancement compared to the baseline approach.

Table 1: Summary of the methods used for workload allocation decision

Ref/ year	Method	Goal	Technology	Performance analysis	Implementation tools
[136]/ 2020	Determine whether to calculate the tasks locally, on an edge server, or in a cloud computing center.	Transmission delay and energy consumption.	Deep reinforcement learning	Achieved the lowest delay compared to the baseline strategy	Simulation - The network consists of 100 sensing devices, 6–20 automobiles, 50 MEC servers, and 1 cloud center.
[134]/ 2020	Determine whether to offload the workload to the edge server or the cloud via WAL or cellular network.	Achieve the lowest response time for the successfully executed task	Machin learning – linear regression	Achieved minimum response time and high task completion compared to other methods	Simulation - EdgeCloudSim simulator
[126]/ 2021	Determine where to deploy the AI workload among cloud server, edge server, or user devices considering AI model complexity, size of the dataset, the network condition, and the computational ability of the cloud, server, and end device.	Minimize response time for latency-sensitive applications.	Deep learning	Minimize the response time by up to 63% compared to the basic approach.	Simulation - TensorFlow, keras and Edge AIBench.
[129]/ 2021	Balances the accuracy with the power constraint of the edge devices by putting a lightweight DNN model in an auxiliary network and a powerful DNN in a principle network.	Given a power limitation, increase the system's inference accuracy.	Deep learning	Improve inference accuracy up to 3.93%	Simulation – python using ResNet model and WRN model with CIFAR-10 and CIFAR-100 datasets, respectively.
[135]/ 2021	The suggested algorithm considers energy use, processing time, data transmission costs, and energy in standby devices. To get better execution times at cheaper costs, the task scheduling selects a cloud or mobile edge computing (MEC) server or local IoT device.	Reduce energy consumption and task processing time.	Heuristic algorithm	Reduced energy usage by up to 51.6% and increased task completion efficiency by up to 86.6%.	Hardware implementation - IoT devices- Arduino Mega MEC - 5 Raspberry Pi 4 Cloud - Intel Xeon Cascade Lake.
[137]/ 2022	Consider a MEC network with multiple wireless devices and one edge server. Each MEC situation is characterized by a specific task weight. In real-time, each wireless device in the network decides whether to perform computations locally or transfer them to an edge server.	By optimizing both offloading decisions and bandwidth allocation simultaneously, the overall system utility of the MEC network can be improved.	Deep learning	The numerical findings demonstrate that DSLO can swiftly adapt to new MEC conditions and only needs a small number of training samples.	Simulation - CVX toolbox in Matlab for dataset creation and DL implementation tool (not mentioned)
[138]/ 2022	Consider N IoT device, one edge server, and one cloud center. Decide whether to execute the task locally on the IoT device or send the task for further processing. If the decision is to offload the task, then the second stage is to decide if to send the task to the edge server or the	Reduce energy consumption and task processing time	Deep Neural Network (DNN) and Deep Meta-learning	The result shows that the DDMTO method outperforms the other earlier strategy, such as MO-BFO, DMRO, and DDTO.	Simulation- Tensor flow and libraries of MLTs

	cloud center.				
[139]/ 2023	Use the Double Deep Q-Network (DDQN) algorithm for edge-cloud (DDQNEC) to decide where to execute the tasks in the edge server or the cloud.	Minimize delay and meet the communication and computation device constraint.	Deep reinforcement learning	DDQNEC outperforms heuristic approaches in terms of resource utilization, task offloading, and task rejection	Simulation-using Python and PyTorch
[140]/ 2023	IoT-edge offloading scenarios Proposes the deep Post-Decision State (PDS)-learning algorithm, a new learning method that combines the PDS-learning method with the traditional DQN.. When the edge network wasn't available or didn't have the processing capacity to perform new activities, the proposed technique chose to use the cloud center instead.	A compromise between high latency and constrained computing power while maintaining data integrity when offloading.	Double deep Q-learning	The proposed method outperforms other methods considered in the work in terms of job failure rate by 5.7%, delay by 4.5%, cost by 4.6%, energy consumption by 3.9%, and computational overhead by 6.1%.	Simulation-using Keras and TensorFlow

VII. Open Issues and Challenges

There are still several obstacles to overcome the integration of edge-cloud systems with AI technology, such as putting DL on edge and execution location of tasks. The location of execution could be on end devices, edge servers, or a mix of end devices, edge servers, and the cloud. Future work must address and consider various difficulties adapting AI with edge clouds and delivering optimum performance. The most recent challenges will be briefly described here.

Problem with optimization: Numerous goals may be accomplished by merging the DL algorithm with the edge cloud. These goals include lowering energy consumption, improving latency, and boosting accuracy. Finding a solution that balances all of these important aspects of the system is an extraordinarily challenging task [141].

Datasets lack: to create an optimal DL system, large and relevant datasets need to guide the application's learning from them. Finding the ideal and appropriate dataset is crucial and regarded as one of the most crucial aspects of a DL system. Creating one's dataset is often a laborious and time-consuming effort. In recent years, several researchers have used transfer learning methodologies. Which focuses on using the dataset from another project for training the system while using the dataset from the target system for the development and testing stages [142], [143].

Diversity of DL algorithms: There are many distinct kinds of DL algorithms, and each has its own set of characteristic twists and turns. When trying to solve a problem using modern cloud computing architectures, it may be difficult for researchers and developers to evaluate which DL algorithm would be the most beneficial. This is due to no predetermined mapping between the DL models and the technology currently available [144].

Privacy and security: The data employed by a DL system is its core component. Data must be transported securely from end devices to edges, and the system must fulfill security criteria such as confidentiality, authorization, and authentication. If any of these four crucial privacy and security components are lacking, the system's effectiveness and accuracy may be compromised. For instance, in a healthcare IoT DL system, modifying patient information might cause a life-risk situation. This vulnerability may present in many other applications [145]–[147].

Hyper-parameters setting: The hyper-parameter settings significantly impact the performance of DL architectures. The configuration of a significant number of hyper-parameters is necessary for DL. Convolution neural network, for instance, requires the setting of several hyper-parameters, including the number of convolutional layers, pooling size, kernel size, activation function type in each of the convolution layers, number of dense layers, number of kernels, number of neurons in each of the dense layer, the number of hidden units in each layer, weight regularisation, batch size, dropout, and learning rate. These hyper-parameters are not the only ones that can be set. Each hyper-parameter must be adjusted for the proposed system, which can take several hours or months in a large network [148].

Energy consumption: Batteries are the typical source of electricity for edge devices. When employed and implemented in these devices, DL algorithms may increase the number of computing processes, increasing the amount of power consumed [149].

Latency time: The time execution of the tasks depends on many factors in the end device-edge–cloud systems, such as task size, the device capabilities, network conditions, and application model complexity. The workload allocation of the tasks must tradeoff between these several metrics to increase the AI application accuracy, especially for critical time applications.

VIII. CONCLUSION

This article presents the state-of-the-art research being conducted currently to integrate EC with AI, enabling the realization of intelligent edges and edge intelligence. In conclusion, as traditional non-AI approaches have limits in dealing with the complex and dynamic environment in EC, which is also regarded as a complex approach that results in computing intensity, AI could enhance and optimize the performance of EC. On the other hand, EC helps the actual implementation of AI by bringing a quicker response time and a more reliable network state. Accurate modeling of the EC process is challenging because of its complexity, dynamics, and large dimensions, even though research on integrating AI with EC has made significant progress. In order to develop effective strategies, it is crucial to create and implement model-free procedures. Also, How to make AI algorithms more efficient with limited compute and energy resources is the key to deploying AI to the edge of the network, which calls for further research and the development of lightweight AI models. The necessity for real-time processing of data generated by end devices unites several example application drivers, including natural language processing, computer vision, network operations, and AR and VR. Techniques were developed for using the distinct structure of DNN models and the geographic proximity of user requests in EC to speed up DL inference across the cloud, edge server, and end devices. The reviewed studies revealed that balancing latency, accuracy, and other performance metrics is crucial. However, many unresolved issues remain, including improving performance, privacy, benchmarking, and resource management.

REFERENCES

[1] G. Premsankar, M. Di Francesco, and T. Taleb, “Edge Computing for the Internet of Things: A Case Study,” *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1275–1284, 2018, doi: 10.1109/jiot.2018.2805263.
[2] A. Mahmood, W. E. Zhang, and Q. Z. Sheng, “Software-defined heterogeneous vehicular networking: The architectural design and open challenges,” *Futur. Internet*, vol. 11, no. 3, 2019, doi: 10.3390/fi11030070.
[3] B. Ji *et al.*, “Survey on the Internet of Vehicles: Network Architectures and Applications,” *IEEE Commun. Stand. Mag.*, vol. 4, no. 1, pp. 34–41, 2020, doi: 10.1109/MCOMSTD.001.1900053.
[4] W. E. Zhang *et al.*, “The 10 Research Topics in the Internet of Things,” *Proc. - 2020 IEEE 6th Int. Conf. Collab. Internet Comput. CIC 2020*, pp. 34–43, 2020, doi: 10.1109/CIC50333.2020.00015.
[5] P. McEnroe ,S. Wang , and M. Liyanage "A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges", *IEEE Internet of Things Journal*, 9(17), pp. 15435–15459, 2022, doi:10.1109/jiot.2022.3176400.
[6] J. Chen and X. Ran, “Deep Learning With Edge Computing: A Review,” *Proc. IEEE*, vol. 107, no. 8, 2019, doi: 10.1109/JPROC.2019.2921977.
[7] W. Yu *et al.*, “A Survey on the Edge Computing for the Internet of Things,” *IEEE Access*, vol. 6, pp. 6900–6919, 2017, doi: 10.1109/ACCESS.2017.2778504.
[8] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016, doi: 10.1109/JIOT.2016.2579198.
[9] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2008, doi: 10.1145/1496091.1496103.
[10] E. Cuervoy *et al.*, “MAUI: Making smartphones last longer with code offload,” *MobiSys’10 - Proc. 8th Int. Conf. Mob. Syst. Appl. Serv.*, pp. 49–62, 2010, doi: 10.1145/1814433.1814441.
[11] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The Case for VM-based Cloudlets in Mobile Computing,” *IEEE pervasive Comput.*, vol. 1, pp. 1–9, 2006.
[12] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” *MCC’12 - Proc. 1st ACM Mob. Cloud Comput. Work.*, pp. 13–15, 2012, doi: 10.1145/2342509.2342513.
[13] S. Yi, Z. Hao, Z. Qin, and Q. Li, “Fog computing: Platform and applications,” *Proc. - 3rd Work. Hot Top. Web Syst. Technol. HotWeb 2015*, pp. 73–78, 2016, doi: 10.1109/HotWeb.2015.22.
[14] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, “Towards wearable cognitive assistance,” *MobiSys 2014 - Proc. 12th Annu. Int. Conf. Mob. Syst. Appl. Serv.*, no. June 2014, pp. 68–81, 2014, doi: 10.1145/2594368.2594383.
[15] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “CloneCloud: Elastic execution between mobile device and cloud,” *EuroSys’11 - Proc. EuroSys 2011 Conf.*, pp. 301–314, 2011, doi: 10.1145/1966445.1966473.
[16] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, “A Survey on Edge Computing Systems and Tools,” *Proc. IEEE*, pp. 1–24, 2019, doi: 10.1109/JPROC.2019.2920341.
[17] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, “Cachier: Edge-Caching for Recognition Applications,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 276–286, 2017, doi: 10.1109/ICDCS.2017.94.
[18] K. Bhardwaj, M. W. Shih, P. Agarwal, A. Gavrilovska, T. Kim, and K. Schwan, “Fast, scalable and secure onloading of edge functions using Airbox,” *Proc. - 1st IEEE/ACM Symp. Edge Comput. SEC 2016*, pp. 14–27, 2016, doi: 10.1109/SEC.2016.15.
[19] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. De Lara, “CloudPath: A multi-Tier cloud computing framework,” *2017 2nd ACM/IEEE Symp. Edge Comput. SEC 2017*, 2017, doi: 10.1145/3132211.3134464.
[20] M. Jang, K. Schwan, K. Bhardwaj, A. Gavrilovska, and A. Avasthi, “Personal clouds: Sharing and integrating networked resources to enhance end user experiences,” *Proc. - IEEE INFOCOM*, pp. 2220–2228, 2014, doi: 10.1109/INFOCOM.2014.6848165.
[21] P. Liu, D. Willis, and S. Banerjee, “ParaDrop: Enabling lightweight multi-tenancy at the network’s extreme edge,” *Proc. - 1st IEEE/ACM Symp. Edge Comput. SEC 2016*, pp. 1–13, 2016, doi: 10.1109/SEC.2016.39.
[22] S. Engineering, “Cloud Computing,” no. August, 2017. [Online]. Available: www.computer.org
[23] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, “SpanEdge: Towards unifying stream processing over central and near-the-

edge data centers," *Proc. - 1st IEEE/ACM Symp. Edge Comput. SEC 2016*, pp. 168–178, 2016, doi: 10.1109/SEC.2016.17.
[24] Q. Zhang, X. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Big data sharing and processing in collaborative edge environment," *Proc. - 4th IEEE Work. Hot Top. Web Syst. Technol. HotWeb 2016*, pp. 20–25, 2016, doi: 10.1109/HotWeb.2016.12.
[25] Z. W. Xu, "Cloud-sea computing systems: Towards thousand-fold improvement in performance per watt for the coming zettabyte era," *J. Comput. Sci. Technol.*, vol. 29, no. 2, pp. 177–181, 2014, doi: 10.1007/s11390-014-1420-2.
[26] P. Zhou, W. Chen, S. Ji, H. Jiang, L. Yu, and D. Wu, "Privacy-preserving Online Task Allocation in Edge-Computing-Enabled Massive Crowdsensing," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7773–7787, 2019, doi: 10.1109/jiot.2019.2903515.
[27] E. I. Gaura, J. Brusey, M. Allen, R. Wilkins, D. Goldsmith, and R. Rednic, "Edge mining the Internet of Things," no. May 2014, 2013, doi: 10.1109/JSEN.2013.2266895.
[28] Z. Xu, W. Liu, J. Huang, C. Yang, J. Lu, and H. Tan, "Artificial Intelligence for Securing IoT Services in Edge Computing : A Survey," *Secur. Commun. Networks*, vol. 1, no. 3, 2020, doi: 10.1155/2020/8872586.
[29] C. Savaglio and G. Fortino, "A Simulation-driven Methodology for IoT Data Mining Based on Edge Computing," *ACM Trans. Internet Technol.*, vol. 21, no. 2, pp. 1–22, 2021.
[30] A. Abeshu and N. Chilamkurti, "Deep Learning: The Frontier for Distributed Attack Detection in Fog-To-Things Computing," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 169–175, 2018, doi: 10.1109/MCOM.2018.1700332.
[31] H. Li, K. Ota, and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, 2018, doi: 10.1109/MNET.2018.1700202.
[32] J. Zhu, Q. Jiang, Y. Shen, C. Qian, F. Xu, and Q. Zhu, "Application of recurrent neural network to mechanical fault diagnosis: a review," *J. Mech. Sci. Technol.*, vol. 36, no. 2, pp. 527–542, 2022, doi: 10.1007/s12206-022-0102-1.
[33] D. Ghimire, D. Kil, and S. Kim, "A Survey on Efficient Convolutional Neural Networks and Hardware Acceleration," *Electronics*, vol. 11, no. 6, p. 945, 2022, doi: 10.3390/electronics11060945.
[34] A. J. Moshayedi, A. S. Roy, A. Kolahdooz, and Y. Shuxin, "Deep Learning Application Pros And Cons Over Algorithm," *EAI Endorsed Trans. AI Robot.*, vol. 1, pp. 1–13, 2022, doi: 10.4108/airo.v1i.19.
[35] Y. Matsuo *et al.*, "Deep learning, reinforcement learning, and world models," *Neural Networks*, vol. 152, pp. 267–275, 2022, doi: 10.1016/j.neunet.2022.03.037.
[36] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations," *Proc. 26th Int. Confer- ence Mach. Learn.*, p. 8, 2009, doi: 10.1145/1553374.1553453.
[37] H. Khelifi, S. Luo, B. Nour, A. Sellami, and H. Moungla, "Bringing Deep Learning at The Edge of Information-Centric Internet of Things," *IEEE Commun. Lett.*, vol. PP, no. XX, p. 1, 2018, doi: 10.1109/LCOMM.2018.2875978.
[38] A. Murali, N. N. Das, S. S. Sukumaran, K. Chandrasekaran, C. Joseph, and J. P. Martin, "Machine Learning Approaches for Resource Allocation in the Cloud: Critical Reflection," *2018 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2018*, pp. 2073–2079, 2018, doi: 10.1109/ICACCI.2018.8554703.
[39] S. Jayaprakash, M. D. Nagarajan, R. P. de Prado, S. Subramanian, and P. B. Divakarachari, "A systematic review of energy management strategies for resource allocation in the cloud: Clustering, optimization and machine learning," *Energies*, vol. 14, no. 17, 2021, doi: 10.3390/en14175322.
[40] "AWS DeepLens – Deep learning enabled video camera for developers - AWS." https://aws.amazon.com/deeplens/ (accessed Mar. 07, 2023).
[41] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2020, doi: 10.4324/9780080519340-12.
[42] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," *Proc. Annu. Int. Conf. Mob. Comput. Networking, MOBICOM*, vol. 2015-Septe, pp. 426–438, 2015, doi: 10.1145/2789168.2790123.
[43] C. C. Hung *et al.*, "VideoEdge: Processing camera streams using hierarchical clusters," *Proc. - 2018 3rd ACM/IEEE Symp. Edge Comput. SEC 2018*, pp. 115–131, 2018, doi: 10.1109/SEC.2018.00016.
[44] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [Review Article]," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, 2018, doi: 10.1109/MCI.2018.2840738.
[45] "Deep Learning for Siri's Voice: On-device Deep Mixture Density Networks for Hybrid Unit Selection Synthesis - Apple Machine Learning Research." https://machinelearning.apple.com/research/siri-voices (accessed Mar. 09, 2023).
[46] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," *2016 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. NAACL HLT 2016 - Proc. Conf.*, pp. 260–270, 2016, doi: 10.18653/v1/n16-1030.
[47] "Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone – Google AI Blog." https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html (accessed Mar. 09, 2023).
[48] J. Rowsell, "Working with multimodality: Rethinking literacy in a digital age," *Work. with Multimodality Rethink. Lit. a Digit. Age*, pp. 1–173, 2013, doi: 10.4324/9780203071953.
[49] "Improve Server Response Time | PageSpeed Insights | Google Developers." https://developers.google.com/speed/docs/insights/Server (accessed Mar. 09, 2023).
[50] "Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant - Apple Machine Learning Research." https://machinelearning.apple.com/research/hey-siri (accessed Mar. 09, 2023).
[51] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, "FastGRNN: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network," *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, no. NeurIPS, pp. 9017–9028, 2018.
[52] A. De' Faveri Tron, "Intrusion Detection with Neural Networks," *Optim. Mach. Learn.*, pp. 201–232, 2022, doi: 10.1002/9781119902881.ch8.
[53] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," no. February, pp. 18–21, 2018, doi: 10.14722/ndss.2018.23204.
[54] S. Chinchali *et al.*, "Cellular network traffic scheduling with deep reinforcement learning," *32nd AAAI Conf. Artif. Intell. AAAI 2018*, pp.

766–774, 2018, doi: 10.1609/aaai.v32i1.11339.
[55] N. Tsikoudis, A. Papadogiannakis, and E. P. Markatos, "LEoNIDS: A Low-Latency and Energy-Efficient Network-Level Intrusion Detection System," *IEEE Trans. Emerg. Top. Comput.*, vol. 4, no. 1, pp. 142–155, 2016, doi: 10.1109/TETC.2014.2369958.
[56] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep Reinforcement Learning for Mobile Edge Caching: Review, New Features, and Open Issues," *IEEE Netw.*, vol. 32, no. 6, pp. 50–57, 2018, doi: 10.1109/MNET.2018.1800109.
[57] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and D. I. Kim, "Distributed Deep Learning at the Edge: A Novel Proactive and Cooperative Caching Framework for Mobile Edge Networks," *IEEE Wirel. Commun. Lett.*, vol. 8, no. 4, pp. 1220–1223, 2019, doi: 10.1109/LWC.2019.2912365.
[58] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "DeepSense: A unified deep learning framework for time-series mobile sensing data processing," *26th Int. World Wide Web Conf. WWW 2017*, pp. 351–360, 2017, doi: 10.1145/3038912.3052577.
[59] W. Ouyang and X. Wang, "Joint deep learning for pedestrian detection," *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 2056–2063, 2013, doi: 10.1109/ICCV.2013.257.
[60] L. Li, K. Ota, and M. Dong, "When Weather Matters: IoT-Based Electrical Load Forecasting for Smart Grid," *IEEE Commun. Mag.*, vol. 55, no. 10, pp. 46–51, 2017, doi: 10.1109/MCOM.2017.1700168.
[61] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, 2018, doi: 10.1109/TCAD.2018.2858384.
[62] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon : Scalable Adaptation of Video Analytics," pp. 253–266, 2018, doi: 10.1145/3230543.3230574.
[63] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018, doi: 10.1109/COMST.2018.2844341.
[64] "Hudson Yards Data | Sidewalk Labs | Data Privacy." https://therealdeal.com/new-york/2019/03/15/hudson-yards-smart-city-or-surveillance-city/ (accessed Mar. 09, 2023).
[65] X. Hou, S. Dey, J. Zhang, and M. Budagavi, "Predictive view generation to enable mobile 360-degree and VR experiences," *VR/AR Netw. 2018 - Proc. 2018 Morning Work. Virtual Real. Augment. Real. Network, Part SIGCOMM 2018*, pp. 20–26, 2018, doi: 10.1145/3229625.3229629.
[66] S. Afzal, J. Chen, and K. K. Ramakrishnan, "Characterization of 360-degree videos," *VR/AR Netw. 2017 - Proc. 2017 Work. Virtual Real. Augment. Real. Network, Part SIGCOMM 2017*, pp. 1–6, 2017, doi: 10.1145/3097895.3097896.
[67] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," *25th Annu. Int. Conf. Mob. Comput. Netw.*, 2019, doi: 10.1145/3300061.3300116.
[68] "Enabling full body AR with Mask R-CNN2Go - Meta Research | Meta Research." https://research.facebook.com/blog/2018/1/enabling-full-body-ar-with-mask-r-cnn2go/ (accessed Mar. 09, 2023).
[69] C. Machover and S. E. Tice, "Virtual Reality - Virtual Reality," *IEEE Comput. Graph. Appl.*, vol. 1, no. January, pp. 15–16, 2014.
[70] Z. Chen *et al.*, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," *2017 2nd ACM/IEEE Symp. Edge Comput. SEC 2017*, 2017, doi: 10.1145/3132211.3134458.
[71] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," *Proc. 12th Annu. Int. Conf. Mob. Syst. Appl. Serv. - MobiSys '14*, no. December, pp. 68–81, 2014, [Online]. Available: http://dl.acm.org/citation.cfm?doid=2594368.2594383
[72] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017, [Online]. Available: http://arxiv.org/abs/1704.04861
[73] W. Liu *et al.*, "SSD: Single shot multibox detector," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.
[74] J. Redmon and A. Farhadi, "Yolo V2.0," *Cvpr2017*, no. April, pp. 187–213, 2016, [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/9789812771728_0012
[75] 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 <0.5MB model size," pp. 1–13, 2016, [Online]. Available: http://arxiv.org/abs/1602.07360
[76] "TensorFlow." https://www.tensorflow.org/ (accessed Mar. 09, 2023).
[77] "Caffe2 and PyTorch join forces to create a Research + Production platform PyTorch 1.0 | Caffe2." https://caffe2.ai/blog/2018/05/02/Caffe2_PyTorch_1_0.html (accessed Mar. 09, 2023).
[78] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pp. 1–14, 2016.
[79] 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," *SenSys 2017 - Proc. 15th ACM Conf. Embed. Networked Sens. Syst.*, vol. 2017-Janua, 2017, doi: 10.1145/3131672.3131675.
[80] L. Lai and N. Suda, "Enabling deep learning at the IoT edge," *IEEE/ACM Int. Conf. Comput. Des. Dig. Tech. Pap. ICCAD*, 2018, doi: 10.1145/3240765.3243473.
[81] S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," *FPGA 2017 - Proc. 2017 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, pp. 75–84, 2017, doi: 10.1145/3020078.3021745.
[82] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," *Proc. 14th ACM Conf. Embed. Networked Sens. Syst. SenSys 2016*, pp. 176–189, 2016, doi: 10.1145/2994551.2994564.
[83] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," pp. 1–9, 2015, [Online]. Available: http://arxiv.org/abs/1503.02531
[84] S. Teerapittayanon, B. McDanel, and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," *Proc. - Int. Conf. Pattern Recognit.*, vol. 0, pp. 2464–2469, 2016, doi: 10.1109/ICPR.2016.7900006.
[85] S. Liu, K. Nan, Y. Lin, H. Liu, Z. Zhou, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," *MobiSys 2018 - Proc. 16th ACM Int. Conf. Mob. Syst. Appl. Serv.*, pp. 389–400, 2018, doi:

10.1145/3210240.3210337.
[86] L. N. Huynh, Y. Lee, and R. K. Balan, "DeepMon: Mobile GPU-based deep learning framework for continuous vision applications," *MobiSys 2017 - Proc. 15th Annu. Int. Conf. Mob. Syst. Appl. Serv.*, pp. 82–95, 2017, doi: 10.1145/3081333.3081360.
[87] "Edge TPU - Run Inference at the Edge | Google Cloud." https://cloud.google.com/edge-tpu/ (accessed Mar. 09, 2023).
[88] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," *Proc. - Int. Symp. Comput. Archit.*, vol. 13-17-June, pp. 92–104, 2015, doi: 10.1145/2749469.2750389.
[89] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "DianNao Family: Energy-Efficient Hardware Accelerators for Machine Learning," *Commun. ACM*, vol. 59, no. 11, pp. 105–112, 2016, doi: 10.1145/2996864.
[90] S. Rivas-Gomez, A. J. Pena, D. Moloney, E. Laure, and S. Markidis, "Exploring the vision processing unit as co-processor for inference," *Proc. - 2018 IEEE 32nd Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2018*, pp. 589–598, 2018, doi: 10.1109/IPDPSW.2018.00098.
[91] "Intel® Movidius^{TM} Vision Processing Units (VPUs)." https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html (accessed Mar. 09, 2023).
[92] "NVIDIA EGX: Accelerating Edge Computing for AI at the Edge | NVIDIA." https://www.nvidia.com/es-la/data-center/data-center-gpus/egx-edge-computing/ (accessed Mar. 09, 2023).
[93] "Qualcomm Neural Processing SDK for AI - Qualcomm Developer Network." https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk (accessed Mar. 09, 2023).
[94] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "RSTensorFlow: GPU enabled TensorFlow for deep learning on commodity android devices," *EMDL 2017 - Proc. 1st Int. Work. Deep Learn. Mob. Syst. Appl. co-located with MobiSys 2017*, pp. 7–12, 2017, doi: 10.1145/3089801.3089805.
[95] N. D. Lane *et al.*, "DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices," *2016 15th ACM/IEEE Int. Conf. Inf. Process. Sens. Networks, IPSN 2016 - Proc.*, no. 1, 2016, doi: 10.1109/IPSN.2016.7460664.
[96] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017, doi: 10.1109/JPROC.2017.2761740.
[97] Q. Wang, G. Jin, Q. Li, K. Wang, Z. Yang, and H. Wang, "Industrial Edge Computing: Vision and Challenges," *Inf. Control*, vol. 50, no. 3, pp. 257–274, 2021, doi: 10.13976/j.cnki.xk.2021.1030.
[98] X. Wang, X. Wang, and S. Mao, "RF Sensing in the Internet of Things: A General Deep Learning Framework," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 62–67, 2018, doi: 10.1109/MCOM.2018.1701277.
[99] T. Y. Chen, "Glimpse : Continuous , Real-Time Object Recognition on Mobile Devices Categories and Subject Descriptors," *SenSys '15 Proc. 13th ACM Conf. Embed. Networked Sens. Syst.*, vol. 15, pp. 155–168, 2015.
[100] C. Liu *et al.*, "A New Deep Learning-Based Food Recognition System for Dietary Assessment on An Edge Computing Service Infrastructure," *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 249–261, 2018, doi: 10.1109/TSC.2017.2662008.
[101] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," *Proc. 14th USENIX Symp. Networked Syst. Des. Implementation, NSDI 2017*, pp. 377–392, 2017.
[102] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," *SIGCOMM 2018 - Proc. 2018 Conf. ACM Spec. Interes. Gr. Data Commun.*, pp. 253–266, 2018, doi: 10.1145/3230543.3230574.
[103] A. H. Jiang *et al.*, "Mainstream : Dynamic Stem-Sharing for Multi-Tenant Video Processing," *2018 USENIX Annu. Tech. Conf. (USENIX ATC '18)*, 2018.
[104] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," *Proc. - IEEE INFOCOM*, vol. 2018-April, pp. 1421–1429, 2018, doi: 10.1109/INFOCOM.2018.8485905.
[105] X. Ran, H. Chen, Z. Liu, and J. Chen, "Delivering deep learning to mobile devices via offloading," *VR/AR Netw. 2017 - Proc. 2017 Work. Virtual Real. Augment. Real. Network, Part SIGCOMM 2017*, pp. 42–47, 2017, doi: 10.1145/3097895.3097903.
[106] 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," *MobiSys 2016 - Proc. 14th Annu. Int. Conf. Mob. Syst. Appl. Serv.*, pp. 123–136, 2016, doi: 10.1145/2906388.2906396.
[107] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: Latency-Aware video analytics on edge computing platform," *2017 2nd ACM/IEEE Symp. Edge Comput. SEC 2017*, 2017, doi: 10.1145/3132211.3134459.
[108] Y. Kang *et al.*, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017, doi: 10.1145/3093337.3037698.
[109] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for Deep Neural Network," *Proc. 2017 Des. Autom. Test Eur. DATE 2017*, pp. 1396–1401, 2017, doi: 10.23919/DATE.2017.7927211.
[110] M. R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," *MobiSys'11 - Compil. Proc. 9th Int. Conf. Mob. Syst. Appl. Serv. Co-located Work.*, pp. 43–56, 2011, doi: 10.1145/1999995.2000000.
[111] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed Deep Neural Networks over the Cloud, the Edge and End Devices," *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 328–339, 2017, doi: 10.1109/ICDCS.2017.226.
[112] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image recognition applications at the edge," *2017 2nd ACM/IEEE Symp. Edge Comput. SEC 2017*, 2017, doi: 10.1145/3132211.3134456.
[113] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed Deep Learning-based Offloading for Mobile Edge Computing Networks," *Mob. Networks Appl.*, vol. 27, no. 3, pp. 1123–1130, 2022, doi: 10.1007/s11036-018-1177-x.
[114] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z. H. Abbas, and S. Khaf, "A deep learning approach for energy efficient computational offloading in mobile edge computing," *IEEE Access*, vol. 7, pp. 149623–149633, 2019, doi: 10.1109/ACCESS.2019.2947053.
[115] Y. Miao, G. Wu, M. Li, A. Ghoneim, M. Al-Rakhami, and M. S. Hossain, "Intelligent task prediction and computation offloading based on mobile-edge cloud computing", Future Generation Computer Systems, 102, pp. 925–931., 2020, doi:10.1016/j.future.2019.09.035.
[116] M. Guo, L. Li, and Q. Guan, "Energy-efficient and delay-guaranteed workload allocation in iot-edge-cloud computing systems," *IEEE Access*, vol. 7, pp. 78685–78697, 2019, doi: 10.1109/ACCESS.2019.2922992.

[117] G. Wang, X. Yang, W. Cai, and Y. Zhang, “Event-triggered online energy flow control strategy for regional integrated energy system using Lyapunov optimization,” *Int. J. Electr. Power Energy Syst.*, vol. 125, no. August 2020, p. 106451, 2021, doi: 10.1016/j.ijepes.2020.106451.
[118] L. Chen, S. Zhou, and J. Xu, “Computation Peer Offloading for Energy-Constrained Mobile Edge Computing in Small-Cell Networks,” *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1932, 2018, doi: 10.1109/TNET.2018.2841758.
[119] C. F. Liu, M. Bennis, M. Debbah, and H. Vincent Poor, “Dynamic Task Offloading and Resource Allocation for Ultra-Reliable Low-Latency Edge Computing,” *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132–4150, 2019, doi: 10.1109/TCOMM.2019.2898573.
[120] Y. H. Chiang, T. Zhang, and Y. Ji, “Joint Cotask-Aware Offloading and Scheduling in Mobile Edge Computing Systems,” *IEEE Access*, vol. 7, pp. 105008–105018, 2019, doi: 10.1109/ACCESS.2019.2931336.
[121] M. Chen and Y. Hao, “Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network,” *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, 2018, doi: 10.1109/JSAC.2018.2815360.
[122] Z. Ning, P. Dong, X. Kong, and F. Xia, “A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4804–4814, 2019, doi: 10.1109/JIOT.2018.2868616.
[123] J. Du, L. Zhao, J. Feng, and X. Chu, “Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems with Min-Max Fairness Guarantee,” *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1594–1608, 2018, doi: 10.1109/TCOMM.2017.2787700.
[124] Y. Wang, X. Tao, X. Zhang, P. Zhang, and Y. T. Hou, “Cooperative Task Offloading in Three-Tier Mobile Computing Networks: An ADMM Framework,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 3, pp. 2763–2776, 2019, doi: 10.1109/TVT.2019.2892176.
[125] Z. Zheng, L. Song, Z. Han, G. Y. Li, and H. V. Poor, “A stackelberg game approach to proactive caching in large-scale mobile edge networks,” *IEEE Trans. Wirel. Commun.*, vol. 17, no. 8, pp. 5198–5211, 2018, doi: 10.1109/TWC.2018.2839111.
[126] T. Hao, J. Zhan, K. Hwang, W. Gao, and X. Wen, “AI-oriented workload allocation for cloud-edge computing,” *Proc. - 21st IEEE/ACM Int. Symp. Clust. Cloud Internet Comput. CCGrid 2021*, pp. 555–564, 2021, doi: 10.1109/CCGrid51090.2021.00065.
[127] T. Hao, J. Zhan, K. Hwang, W. Gao, and X. Wen, “AI-oriented Medical Workload Allocation for Hierarchical Cloud/Edge/Device Computing,” 2020, [Online]. Available: http://arxiv.org/abs/2002.03493
[128] T. Hao *et al.*, “Edge AIBench: Towards Comprehensive End-to-End Edge Computing Benchmarking,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11459 LNCS, pp. 23–30, 2019, doi: 10.1007/978-3-030-32813-9_3.
[129] Y. W. Hung, Y. C. Chen, C. Lo, A. G. So, and S. C. Chang, “Dynamic Workload Allocation for Edge Computing,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 29, no. 3, pp. 519–529, 2021, doi: 10.1109/TVLSI.2021.3049520.
[130] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.
[131] S. Zagoruyko and N. Komodakis, “Wide Residual Networks,” *Br. Mach. Vis. Conf. 2016, BMVC 2016*, vol. 2016-Septe, pp. 87.1-87.12, 2016, doi: 10.5244/C.30.87.
[132] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” in *Asha*, 2009.
[133] P. Panda, A. Sengupta, and K. Roy, “Energy-efficient and improved image recognition with conditional deep learning,” *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, pp. 1–21, 2017, doi: 10.1145/3007192.
[134] C. Sonmez, C. Tunca, A. Ozgovde, and C. Ersoy, “Machine Learning-Based Workload Orchestrator for Vehicular Edge Computing,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2239–2251, 2020, doi: 10.1109/tits.2020.3024233.
[135] J. C. S. Dos Anjos, J. L. G. Gross, K. J. Matteussi, G. V. González, V. R. Q. Leithardt, and C. F. R. Geyer, “An algorithm to minimize energy consumption and elapsed time for iot workloads in a hybrid architecture,” *Sensors*, vol. 21, no. 9, pp. 1–20, 2021, doi: 10.3390/s21092914.
[136] J. Long, Y. Luo, X. Zhu, E. Luo, and M. Huang, “Computation offloading through mobile vehicles in IoT-edge-cloud network,” *Eurasip J. Wirel. Commun. Netw.*, vol. 2020, no. 1, 2020, doi: 10.1186/s13638-020-01848-5.
[137] S. Yang, G. Lee, and L. Huang, “Deep Learning-Based Dynamic Computation Task Offloading for Mobile Edge Computing Networks,” *Sensors*, vol. 22, no. 11, pp. 1–18, 2022, doi: 10.3390/s22114088.
[138] M. Gali and A. Mahamkali, “A Distributed Deep Meta Learning based Task Offloading Framework for Smart City Internet of Things with Edge-Cloud Computing,” vol. 4, pp. 224–237, 2022, doi: 10.58346/JISIS.2022.I4.016.
[139] I. Ullah, H.-K. Lim, Y. J. Seok, and Y.-H. Han, “Optimizing Task Offloading and Resource Allocation in Edge-Cloud Networks : A DRL Approach,” pp. 1–28, 2023, doi: 10.21203/rs.3.rs-2522525/v1.
[140] A. Heidari, N. Jafari, M. Ali, and J. Jamali, “A green , secure , and deep intelligent method for dynamic IoT-edge-cloud offloading scenarios,” *Sustain. Comput. Informatics Syst.*, vol. 38, no. February, p. 100859, 2023, doi: 10.1016/j.suscom.2023.100859.
[141] J. Chen and X. Ran, “Deep Learning With Edge Computing: A Review,” *Proc. IEEE*, pp. 1–19, 2019, doi: 10.1109/JPROC.2019.2921977.
[142] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi, and K. L. Tan, “Database meets deep learning: Challenges and opportunities,” *SIGMOD Rec.*, vol. 45, no. 2, pp. 17–22, 2016, doi: 10.1145/3003665.3003669.
[143] I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. W. Aha, “Unsupervised and transfer learning challenge,” *Proc. Int. Jt. Conf. Neural Networks*, pp. 793–800, 2011, doi: 10.1109/IJCNN.2011.6033302.
[144] X. Chen, S. Member, and X. Lin, “Big Data Deep Learning : Challenges and Perspectives,” *IEEE Access*, vol. 2, pp. 514–525, 2014, doi: 10.1109/ACCESS.2014.2325029.
[145] A. Mohammad and M. Pradhan, “Machine learning with big data analytics for cloud security,” *Comput. &amp; Electr. Eng.*, vol. 96, p. 107527, 2021, doi: 10.1016/j.compeleceng.2021.107527.
[146] U. A. Butt *et al.*, “A review of machine learning algorithms for cloud computing security,” *Electron.*, vol. 9, no. 9, pp. 1–25, 2020, doi: 10.3390/electronics9091379.
[147] H. Wu, X. Li, and Y. Deng, “Deep learning-driven wireless communication for edge-cloud computing: opportunities and challenges,” *J. Cloud Comput.*, vol. 9, no. 1, 2020, doi: 10.1186/s13677-020-00168-9.
[148] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, “Machine Learning Meets Computation and Communication Control in Evolving Edge and Cloud: Challenges and Future Perspective,” *IEEE Commun. Surv. Tutorials*, vol. 22, no. 1, pp. 38–67, 2020, doi: 10.1109/COMST.2019.2943405.
[149] H.-A. Ounifi, A. Gherbi, and N. Kara, “Deep machine learning-based power usage effectiveness prediction for sustainable cloud

infrastructures," *Sustain. Energy Technol. Assessments*, vol. 52, p. 101967, 2022, doi: 10.1016/j.seta.2022.101967.