Towards Optimal Placement of Deep Learning Tasks Over Edge and Cloud Infrastructures

Anonymous authors
Paper under double-blind review

Abstract

Edge intelligent applications like VR/AR and surveillance have become popular with the growth of IoT and mobile devices. However, edge devices with limited capacity struggle to serve increasingly large and complex deep learning (DL) models. To mitigate such challenges, researchers have proposed optimizing and offloading partitions of DL models among user devices, edge servers, and the cloud. In this setting, users can take advantage of different services to support their intelligent applications. For example, edge resources offer low response latency. In contrast, cloud platforms provide low monetary cost computation resources for computation-intensive workloads. However, communication between DL model partitions can introduce transmission bottlenecks and pose risks of data leakage. Recent research aims to balance accuracy, computation delay, transmission delay, and privacy concerns. They address these issues with model compression, model distillation, transmission compression, and model architecture adaptations, including internal classifiers. This survey develops a systematic evaluation approach for state-of-the-art model offloading methods and model adaptation techniques. We formulate an optimization problem for edge Deep Neural Network offloading that optimizes inference and training latency, data privacy, and resource monetary cost.

1 Introduction

In recent decades, rich data have been generated on mobile or IoT devices. Computing resources in the cloud remain more flexible in scaling and management than those on the edge. However, edge computing can mitigate the transmission bottleneck [16]. Recent intelligent systems focus on offloading user applications to the multi-layer cloud, edge, and personal device systems [68, 93, 95], leveraging the computation capacity of powerful edge servers or the core cloud while improving latency, privacy, and monetary cost efficiency.

Machine Learning (ML) applications experience heterogeneous performance requirements and resource availability. Mobile and IoT applications, for example object recognition in housekeeping AIoT devices [133] and localization of autonomous cars [21], are constrained by energy consumption [71] or are based on emerging infrastructures such as 5G/6G base stations and smart cities [38, 140, 142].

Meanwhile, the high monetary cost associated with computational resources for AI/ML training and serving poses a barrier, particularly for research institutions and smaller companies. This is in contrast to large companies which have the engineering capacity to build Deep Learning (DL) clusters. For example, pre-training LLaMA-3.1-8B (LLaMA-3.1-405B) requires 1.46 million (30.84 million) GPU hours on H100-80GB GPUs [107], and ByteDance operates a cluster of over $10,000$ NVIDIA Ampere GPUs for their Large Language Model (LLM) workloads [73].

As AI/ML applications become popular, building a monetary cost ($) efficient, latency optimized, and privacy-aware infrastructure for architecture-optimized models has emerged as a critical area of research [44, 51, 158]. This survey focuses on the computation offloading problem between the edge and cloud. We also discuss orthogonal methods to improve latency, privacy and monetary cost of ML systems in addition to computation offloading, including quantization [114], weight pruning [91], distillation [47], privacy preserving distributed DL [162] and cost ($) based resource provisioning using cloud services [30].

DL applications can be decomposed by splitting the layers of neural networks (NN) and offloading the model partitions to cloud, edge, and client mobile devices or IoT devices. By considering each partition of an NN as a general operator, we believe that insights of this survey also apply to general application decomposition problems.

Machine Learning as a Service. Machine Learning as a Service (MLaaS) is a type of cloud service with a cost-effective cost model that abstracts the computation resources for ML model training and inference, so that the user only pays for the resources that their models leverage. Such systems face challenges in adapting to LLM training and inference paradigms, which require massive computation resources and inter-node communication bandwidth often found in private GPU clusters [73]. However, MLaaS remains an important service for many LLM tasks. Compared to pre-training an LLM, parameter-efficient fine-tuning and inference on smaller models are more cost-effective. For example, fine-tuning a Low-Rank Adapter (LoRA) for a model with $65B$ weights requires only a single $48GB$ GPU for less than 24 hours [31]. Additionally, running inference on a $Qwen2-7B-Instruct$ model using one $A100-80GB$ GPU achieves 41.20 Tokens per Second [121]. Therefore, instead of constructing their own cluster, small-size companies can fine-tune or serve the components of decomposed models, such as LoRA adapters for LLM or traditional deep neural networks, by acting as a broker offering a Machine Learning as a Service (MLaaS) cost model to meet the demand of their customers and for internal R&D.

Recently, organizations like Adobe [19, 22] and Workday [161], have built their MLaaS platforms by provisioning cloud resources, including Virtual Machines (VMs), containers, etc., from cloud service providers, such as Amazon Web Service (AWS), Microsoft Azure, and Google Cloud Platform (GCP), while maintaining a private cloud cluster. They then built their own ML services using that hybrid cloud infrastructure. This approach presents tradeoffs and opportunities in service latency guarantee, user data privacy preservation, and saving in monetary cost of computation resources.

For example, provisioning $H100$ GPUs in the cloud enables low processing time, but results in a high monetary resource cost ($) and potentially increases data transmission time compared with edge resources. Similarly, allocating computation resources in the cloud instead of processing data securely at the edge, i.e., private cloud or user devices, can minimize the monetary cost of resource maintenance ($) and processing delay, but increases the risk of data leakage.

Furthermore, some service level objectives (accuracy, latency, privacy, and monetary cost ($)) can be relaxed to improve other service level objectives. For example, a software vulnerability detection system for a data center has low sensitivity to the privacy of the software discovered when reporting to the cluster administrator. Thus, an MLaaS broker can take advantage of the high parallelism available in the cloud for inference or training without incurring significant monetary costs of computational resources [178]. Furthermore, agreement-based user data sharing can further mitigate user data privacy concerns [46, 118].

Investigating such tradeoffs even with traditional DNNs as opposed to LLMs provides valuable insights for the future design of MLaaS systems and for supporting large ML systems with short latency, high privacy guarantee and low monetary cost of resources for smaller companies and institutions. Recent research has focused on computation-efficient and privacy-aware ML models or cost-efficient resource orchestration methods. However, a study of the interactions between all three aspects of latency, privacy, and monetary cost remains an ongoing topic. In this survey, we explore recent works and open issues surrounding these interactions via resource and model adaptations.

What are the limitations of existing MLaaS systems? Existing MLaaS systems provide managed services in the cloud and on the edge. Depending on their flexibility in configuration, MLaaS systems manage different resources for users at various levels of abstraction [171]. AWS provides a set of *AI services*, including Amazon Rekognition [12], which hide lower-level details like machine learning (ML) models and computation resources from users. On the other hand, they also offer *ML services*. For example, Amazon Sagemaker [13] allows users to define models, data sources, and resource orchestration across Virtual Machine (VM) instances, serverless instances, S3 object storage, etc. Sagemaker Edge [14] can deploy an NN model and collect data on edge IoT devices owned by the user for ML inference and model retraining. However, these services do not incorporate much of the existing research in model adaptations for optimal latency [147], privacy [116], and service cost [32, 165]. Sagemaker Edge compiles NN models to utilize the client's hardware architecture and memory access patterns for optimal ML inference and training speed [15], which is a small subset of model adaptation.

Therefore, this survey investigates model architecture optimization and ML task resource provisioning strategies that could become part of future MLaaS services, empowering users to develop their applications in the cloud or at the edge. We particularly study model offloading and model adaptation techniques.

Why offloading DL tasks? ML applications that collect large volumes of data and employ highly parameterized deep learning models usually face a short latency requirement for Quality of Service [52, 64, 99, 153]. As shown in Table 1, recent lightweight DNN models used in Augmented Reality (AR) applications running on edge devices, such as Raspberry Pi or mobile phones, often struggle to meet the 30 FPS video requirements or 100ms human-sensible end-to-end (frame refresh) latency target [28, 110]. Due to limitations in computation capacity [39, 128], bandwidth [106], battery capacity [80], and memory/storage space [98], edge devices cannot sustain high performance.

Cloud resources offer alternatives to edge computing with extra processing capacity, as shown in Table 2. However, the public cloud faces privacy concerns [179] and transmission bottlenecks over the Internet [39, 181], which prohibit transmitting source data to the cloud for various ML tasks.

To take advantage of the strengths of both edge and cloud platforms, recent research has focused on partially offloading computation from the cloud to the edge while securing customer data at edge servers or user devices. In this approach, a portion of the computation is performed on the client devices. Only the essential hidden variables required to complete the inference or training tasks at high accuracy are transmitted to the cloud or edge server. This paradigm keeps the source data on the client device, enhancing the efficiency and privacy of transmission. Since data is only sent to remote servers when necessary, this approach can reduce overall ML job completion latency. As shown in Table 3, some existing studies have shown low latency and high model accuracy.

Challenges of DL task offloading. Finding an optimal offloading plan for DL applications is not trivial. Given a Neural Network (NN) model and a data source, the model can

Model	Device	End-to-End Latency	Power	Pred Metrics	Dataset/Task	Citation
MobileNetV3 [57]	Raspberry Pi $4B+$	$595ms$	NA	79.23% accuracy	$48*48$-pixel RAF-DB [90]	[64]
MobileNetV2 [134]	Raspberry Pi $4B+$	$3571ms$	NA	81.16% accuracy	$48*48$-pixel RAF-DB [90]	[64]
MobileNetV2+SSDLite [134]	Google Pixel 1	$162ms$	NA	22.1% mAP	COCO [94]	[57]
MobileNetV3+SSDLite [134]	Google Pixel 1	$137ms$	NA	22.0% mAP	COCO [94]	[57]
YOLO(YOLOv3) [126]	Google Pixel 2	$4500ms$	$4.4W$	40% IOU	Imagenet Video [79]	[24]
Tiny-YOLO(YOLOv2) [125]	Google Pixel 2	$1200ms$	$4W$	40% IOU	[79]	[24]

Table 1: DNN performances at edge in recent works

Model	Hardware	Processing Latency	Pred Metrics	Dataset/Task	Citation
YOLOv4-608 [126]	Tesla V100	$16.1ms$	43.5% COCOmAP	COCO [94]	[25]
YOLOv3-608 [126]	Nvidia Titan X	$57.9ms$	33% COCOmAP	COCO [94]	[126]
YOLOv2-544 [125]	Nvidia Titan X	NA	21.6% COCOmAP	COCO [94]	[126]

Table 2: DNN performances at cloud in recent works

Model	Edge	Cloud	End-to-End Latency	Pred Metrics	Bandwidth	Dataset/Task	Citation
Faster R-CNN (ResNet-50) [127]	NV Jetson TX2	NV Titan XP	$34.56ms$	70% IoU	82.8Mbps	Object detection Xiph dataset [11]	Baseline [100]
Faster R-CNN (ResNet-50) [127]	NV Jetson TX2	NV Titan XP	$22.96ms$	75.8% IoU	276Mbps	Object detection Xiph dataset [11]	Baseline [100]
Faster R-CNN (ResNet-50) [127]	NV Jetson TX2	NV Titan XP	$17.23ms$	86.4% IoU	82.8Mbps	Object detection Xiph dataset [11]	DRE+PSI +MvOT [100]
Faster R-CNN (ResNet-50) [127]	NV Jetson TX2	NV Titan XP	$15.52ms$	91.1% IoU	276Mbps	Object detection Xiph dataset [11]	DRE+PSI +MvOT [100]

Table 3: End-to-end DNN performances combining edge and cloud in recent works

be partitioned and deployed on the cloud, edge server, or client devices. However, a naïve offloading plan can result in long transmission and processing delays, privacy breaches, or resource under-provisioning and over-provisioning. In this survey, we formulate an optimization problem trading off optimization objectives, including Latency, Privacy, and Monetary Cost ($), based on various existing methods. Previous surveys have addressed aspects of optimizing monetary cost, latency or privacy for AI applications (Table 4). However, they do not formulate the optimization problem nor discuss monetary cost ($) based approaches. Detailed cost analysis using real-world cloud resources for low-cost ($) ML serving and training remains limited. Furthermore, while some existing surveys [105, 158] provide valuable insights, they often lack comprehensive discussions on source data privacy in distributed inference and training systems. To address this gap, our survey highlights recent work addressing model inversion attacks [41, 156, 169].

We organize the paper based on optimization objectives. In Sec. 2, we introduce the optimization problem by studying the challenges of ML task offloading given different optimization objectives, including Latency in Sec. 2.1.1, Privacy in Sec. 2.1.2, and Monetary Cost ($) in Sec. 2.1.3. Then, we formulate the optimization problem for Latency (Sec. 2.2.1), Privacy (Sec. 2.2.2) and Monetary Cost (Sec. 2.2.3). In Sec. 3, we discuss popular adaptive learning methods to deploy a DNN model across the spectrum of cloud, edge, and client resources by optimizing Latency (Sec. 3.1), Privacy (Sec. 3.2), and Monetary Cost (Sec. 3.3). Sec. 4 concludes the paper.

2 Problem Definition

Recent studies have explored offloading a Deep Neural Network (DNN) model, both training and inference jobs, across the core cloud, edge, and client devices to meet resource constraints and privacy guarantees. With environment dynamics, each inference request can adaptively go through model partitions using the most capable resources to minimize latency and meet privacy guarantees constrained on other performance requirements. Similarly, for training jobs, although all partitions should participate in the training, using the most capable resources also minimizes training dataset transmission delay and training time, while enhancing data privacy by limiting the exposure of sensitive information. However, partitioning the NN model introduces new challenges. Between model partitions, hidden variables and gradients transmit-

Optimization Formulation	Monetary Cost ($)	Latency	Privacy	DL	Placement	Scope	Inference	Training	Reference
✓	✓	✓	✓	✓	✓	Edge Devices & Edge & Cloud	✓	✓	(Our Work)
✗	✓	✓	✗	✓	✓	Graph in Mobile & Cloud	✓	✓	2020 [154]
✗	✗	✓	✓	✓	✓	AIoT & Edge & Cloud	✓	✓	2021 [26]
✗	✗	✓	◦	✓	✓	Early Exit in Mobile & Cloud	✓	✓	2022 [105]
✗	✗	✓	✓	✓	✓	End Device & Edge & Cloud	✓	✓	2023 [35]
✗	✗	✓	◦	✓	✓	End Device & Edge & Cloud	✓	✓	2024 [158]

Table 4: Related survey comparison: ✓indicates the corresponding survey covers up-to-date or more comprehensive discussion. ◦ indicates our work is more complementary or has different discussion than the corresponding work. ✗means the corresponding work does not discuss this aspect.

ted during forward and backward propagation add additional transmission overhead [39, 170, 181] and cause client data leakage [150, 179]. Meanwhile, byproducts of running on the edge, for example, extra processing delays [61, 104] and energy consumption [75], should be minimized. In this section, we discuss model offloading challenges (Sec. 2.1) and problem definitions over cloud, edge, and resource-constrained client devices (Sec. 2.2).

2.1 DNN Offloading Challenges

Existing MLaaS systems manage cloud resources [12] or user devices to run DL jobs [14]. Meanwhile, cloud-managed edge computing resources, including AWS Local Zones [5] and Wavelength [6], and edge ML model optimizer have become important building blocks for ML services used by companies such as Holo-Light [60], Netflix [115], and SKT [143], etc. With AWS Sagemaker Edge [14] and AWS Greengrass [4], a user can optimize their edge application by a compilation that targets their specific hardware (CPU architecture) and operating system. In the future, we envision MLaaS service providers adopting more model and resource adaptations in their optimizers, improving *latency* of processing and transmission, *privacy* of the source data, and the monetary cost of resources. To enable such optimizers, we study the challenges of achieving high DNN performance when a DL model is partitioned between cloud, edge, and user devices.

2.1.1 Latency

The time spent in a distributed ML training or inference system can be decomposed into transmission and processing delays. When large volumes of data are sent between model partitions, transmission overhead can dominate training or inference latency [98, 179]. Meanwhile, offloading too many

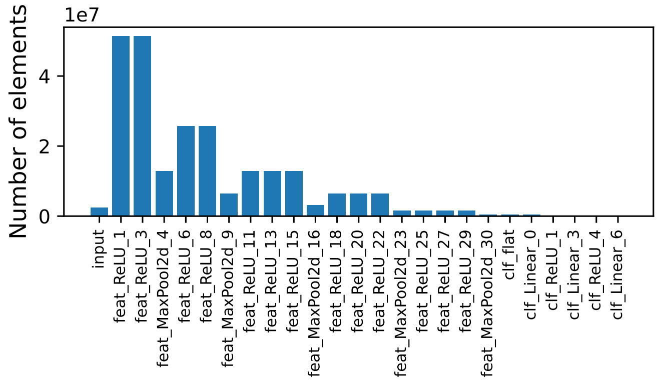

Figure 1: Hidden variable sizes of VGG16 with CiFAR-10 and batch size of 16.

model parameters to constrained edge resources can also overwhelm user devices, resulting in long processing delays. Ideally, a practical NN partitioning paradigm should optimize for both delays to ensure optimal latency performance.

Transmission. For a partitioned NN, hidden variables (or activations)[1] and gradients must be sent between partitions to complete forward and backward propagation, often over the internet limited by bandwidth in IoT or mobile device-based systems. In Fig. 1, we profiled the hidden variable sizes using a VGG-16 model [141] and CiFAR-10 [78] with a batch size of 16. The x-axis indicates the NN layer where the model is split, where the head part of the DNN (from the input layer up to and including the splitting layer) runs on a client device, and the tail part runs on an edge or cloud server. The y-axis shows the output size of different splitting layers. Different splitting layers yield different output sizes. Therefore, the model splitting can be optimized for short delays [75].

[1]We use the terms "hidden variables" and "activations" interchangeably.

Furthermore, previous work has explored the use of intermediate data and model compression methods to reduce communication overhead. By incorporating a bottleneck network, such as Auto-Encoder, at the splitting point, previous works select the key features for transmission [39, 62, 63, 104, 138, 170, 181]. On the other hand, model trimming techniques, such as model distillation [56] and quantization [61, 66, 101, 135], can also minimize the size of hidden variables (activations) and gradients to send.

The sparse model activates a subset of the model parameters during inference and training. For example, the Internal Classifier (IC) allows forward propagation to end in one partition, and no intermediate data transmission [81, 147]. When a classifier gains confidence in the prediction, it emits the output, and no subsequent feature extraction is needed.

Processing. Another challenge for offloaded deep learning systems is the limited processing capacity of edge and client devices. As shown in Table 1, edge devices such as Raspberry Pi [64] and mobile phones [24, 57] often struggle to meet the latency or accuracy requirements demanded by machine learning applications.

To overcome these limitations, related work has explored model adaptations, including quantization [91], pruning [53, 96, 159, 168], and knowledge distillation [56]. Such methods reduce model weights, allowing applications deployed on edge devices or the cloud to meet QoS requirements.

In addition, other works have explored the use of cloud computing capacity to assist edge intelligence applications. However, this approach introduces challenges in privacy and transmission [62, 175].

2.1.2 Privacy

Privacy of source data has become a critical concern for DL systems. Partitioning and offloading an NN to edge devices helps keep raw source data private, as user data is not sent over the network. However, data breaches can still occur, as adversaries can exploit information in intermediate data through model inversion attacks [41, 156, 169].

Recent works [103, 150, 175, 179] discuss the use of Auto-Encoders [17, 131] to reconstruct the source data from the intermediate data sent from the edge to the cloud during forward propagation. The Auto-Encoder consists of an encoder neural network (NN) and a decoder NN, and uses a loss function, e.g., Mean Squared Error (MSE), to gauge the error between the source data and reconstructed data. The encoder mirrors the architecture of the NN on client devices, while the decoder reflects the encoder structure.

One can then train an Auto-Encoder with a dataset similar to the private source data to reconstruct the client source data from activations sent by clients. For example, trained with KMNIST, the decoder NN could reconstruct MNIST hand-writings [179]. Or, with generic facial images from the Internet, a trained decoder NN can reconstruct the source data of a facial recognition classifier [41].

Such an attack is practical in real-world settings. First, the decoder structure can be flexible and it is not required to precisely mirror the client model [89, 179]. Second, many recent large-scale DL training systems are variants of Federated Learning schemes [89, 106] where multiple computing nodes simultaneously train local models using their private dataset and then aggregate and share their model weights. An honest but curious node can exploit these shared weights to build an Auto-Encoder [89] to reconstruct the source data of other nodes.

Privacy-preserving methods for model and user data are critical for an MLaaS system. Recent work has focused on training privacy-preserving models. One approach involves encryption [74, 113], which, however, can cause significant slowdown [33, 111]. As shown in Table 5, without an encryption method, using Nvidia Titan Xp, each inference with the ImageNet dataset and the VGG-16 network achieves 14.5ms. However, with FALCON encryption, using CPUs, previous work reported 12,960ms for the same task [111].

Another privacy-preserving approach introduces a secondary loss function, e.g., distance correlation [70, 150, 179], to constrain the similarity between intermediate and source data during model training. Similar works also incorporate an Auto-Encoder to model training, using reconstruction error as privacy metric [89]. Furthermore, previous works utilize DNN pruning with *masks* to remove mutual information between the source and intermediate data [33, 111].

Other privacy-preserving methods apply perturbations to intermediate data [103, 175]. Thus, given the adversarial objective, the intermediate data retain minimal sensitive information, while the cloud NN learns to extract key elements for model inference or training.

2.1.3 Monetary Cost

As cloud infrastructure has evolved over the past decades, there are other cloud services other than VMs that are more cost-efficient. For example, different cloud services from different providers, have individualized cost models [119] and various accuracy performances [165]. For example, there are Function-as-a-Service (FaaS) and Container-as-a-Service (CaaS) clusters in the cloud [1, 2] and AWS Lambda@Edge and Local Zones at the edge [3, 5] that offer a fine-grained monetary resource cost ($) and low latency [137]. An MLaaS system should adaptively configure the cost-efficient runtime environment and model architecture for its ML jobs.

For computation-intensive training jobs, achieving cost-efficiency requires designing models and training paradigms that minimize the overall computational burden. Previous works have explored minimizing training rounds before convergence by strategically selecting key data samples, or reducing the per-update computational cost using parameter-efficient fine-tuning (PEFT) techniques such as LoRA. Ex-

Model	Hardware	Processing Latency	Dataset/Task	Privacy	Source
VGG-16 [151]	Nvidia P100	$57ms$	Tiny ImageNet [82]	Plaintext	[151]
VGG-16 [151]	CPU	$1,300ms$	Tiny ImageNet [82]	Plaintext	[151]
VGG-16 [151]	CPU(Local Area Network)	$40,000ms$	Tiny ImageNet [82]	SMPC(FALCON)	[151]
VGG-16 [151]	CPU(Local Area Network)	$59,000ms$	Tiny ImageNet [82]	SMPC(FALCON)	[151]
VGG-16 [111]	Nvidia Titan Xp	$14.5ms$	ImageNet [132]	Plaintext	[111]
VGG-16 [111]	CPU	$12,960ms$	ImageNet [132]	SMPC(FALCON)	[111]
VGG-16 [111]	Nvidia Titan Xp	$14.5ms$	ImageNet [132]	Plaintext(Cloak)	[111]

Table 5: Privacy-preserving DNN inference performances in the core cloud in recent works

amples include methods for Federated Learning (FL) client selection to balance non-IID client data [155] and the application of PEFT within FL settings [177].

More recent research also proposes a Split Federated Learning (SFL) paradigm [148] where under FL setting, each node participating in the training can be offloaded from the cloud to the edge. They explore the transmission and computation demand given different model offloading and aggregation algorithms leveraging individual cost models of resources at the edge and cloud. FSL [179] proposes an offloading approach that tradeoffs transmission and processing delays, privacy and accuracy with different strategies. Other work [97, 119, 149, 160] focuses on resource cost models and tradeoff monetary cost and training time when offloading model partitions.

For highly dynamic inference workloads, slow scaling in the core cloud might result in under or overprovisioning of resources and consequently missing the QoS target or wasting the monetary cost of resources [123, 124]. Related works have explored dynamically directing workload to a deep NN in the cloud and a shallow NN at the edge for cost savings [32]. Other works deploy NN partitions using Function-as-a-Service (FaaS) [69]. This approach leverages the pay-per-use nature of FaaS, where the user only pays for the actual computation time used, to avoid the costs of keeping VMs constantly running and provisioned, including node cold start and model loading time.

Furthermore, specific adaptations to the NN architecture can enable resource provisioning for individual NN layers, achieving cost-effective QoS tracking. By incorporating internal classifiers [76,157] or neuron skipping methods [77], only a subset of the network's neurons is used for prediction. Thus, users can minimize the monetary resource cost ($) based on different cloud resource pricing models [124]. Specifically, low-workload layers can be provisioned on demand with FaaS platforms [2,3,137] without relying on reserved VMs, so there is less idle time for computation resources. Such adaptations can be applied across different ML tasks. For example, in an image classification task, the shallow layers might capture the contour of a banana, while the deep layers that focus on the details of the banana are less critical to some classifications [76, 111]. Consequently, these less frequently used deep layers are well-suited for FaaS.

2.2 Problem Formulation

We devise a deep learning (DL) model adaptation and resource provisioning problem formulation by integrating three sub-formulations to partition the model between edge and cloud resources. The formulation addresses the challenges outlined in the previous section, including

- Challenge 1: To achieve overall short training or inference latency, an NN partition strategy should balance processing and transmission delays.
- Challenge 2: An adversary can use an Auto-Encoder NN to reconstruct source data from hidden variables, which introduces data leakage concerns.
- Challenge 3: We need a fine-grained resource provisioning approach that fits the model architecture to save monetary cost while keeping track of Service Level Objectives.

Specifically, we minimize a weighted sum of loss functions, including Latency (L_l), Privacy(L_p), and Cost (L_c), as a constrained multi-objective optimization problem below.

$$min(w^L \mathcal{L}^L + w^P \mathcal{L}^P + w^C \mathcal{L}^C) \quad (1)$$

$$s.t. \text{ constraints on training loss and inference accuracy.} \quad (2)$$

Constraints for each objective are discussed in Sec. 2.2.1, 2.2.2 and 2.2.3. We illustrate the solutions in Sec. 3.

2.2.1 Latency(L_l)

Balancing and minimizing transmission and processing delays are essential to DL training and inference tasks. Arbitrary model partitioning can cause excessive data transmission. In contrast, deploying too many layers on computation-limited edge devices yields a long processing time.

We focus on a DL model composed of M partitions ($F_{pid}, pid \in 1,2,...,M$ in Fig. 2) with the notations defined in Table 6. An individual model partition pid can be offloaded to the edge or cloud based on the estimation of its training or inference time (denoted by T_{pid}, which is the sum of transmission delay T^T_{pid} and computation delay T^C_{pid}), the hidden variable size ($Size(.)$) and the profiles of floating point operations performed by layers in the partition ($FLOPs^j_i(.)$) In practice, however, from shallow to deep NN partitions, when the offloading decision of one partition changes, the same decision follows for all subsequent partitions to minimize the transmission delay [63, 75, 147].

A model partition can be adapted to reduce inference time (π_{pid} and π_{pid+1} both sides of the dashed line in Fig. 2). Each partition pid can be adapted by attaching Q_{pid} internal classifiers, where each classifier c has the confidence threshold α^c_{pid}, request exit rate β^c_{pid}, and the observed test metrics $A^c_{\pi_{pid}}$, including precision, and recall [61, 81, 86]. We denote the sum of the exit percentages for partition pid as β_{pid}.

Partitions can also be adapted with transmission and model compression methods. For each partition pid, we denote by x_{pid} the hidden variable output (x_0 refers to the source data) and two model compression ratios: (1) γ_{pid} for latent space compression layers (in Fig. 2, dark blue layers represent an encoder and dark orange layers represent the decoder), and (2) κ_{pid} for model compression like knowledge distillation, neuron pruning and quantization [62, 88, 104, 117, 138, 170] as exemplified by light blue and light orange layers in Fig. 2.

We formulate a constrained multi-objective optimization problem for processing and transmission time.

$$\mathcal{L}^L = \min_{pid, \alpha^c_{pid}, \kappa_{pid}, \gamma_{pid}} (\xi^T_0 T^T_0 + \sum_{pid=1}^{M} T_{pid}) \tag{1}$$

$$s.t. \sum_{pid=1}^{M} \sum_{c=1}^{Q_{pid}} \beta^c_{pid} * A^c_{\pi_{pid}} \geq A_{tar} \tag{2}$$

$$\beta_{pid} = \sum_{c=1}^{Q_{pid}} Pr(\alpha'^c_{pid} > \alpha^c_{pid}) = Flag_{Train} * \sum_{c=1}^{Q_{pid}} \beta^c_{pid} \tag{3}$$

$$x_{pid} = \pi_{pid}(F_{pid})(x_{pid-1}) \tag{4}$$

$$T^T_{pid} = \frac{(1-\kappa_{pid})(1-\beta_{pid})(1-\gamma_{pid})Size(F_{pid}(x_{pid-1}))}{bandwidth} \tag{5}$$

$$T^T_0 = \frac{(1-\gamma_0)Size(x_0)}{bandwidth} \tag{6}$$

$$T^C_{pid} = \frac{FLOPs^{pid}_{pid}(x_{pid-1}, \pi_{pid})}{\mu_{pid}} \tag{7}$$

$$T_{pid} = \xi^C_{pid} T^C_{pid} + \xi^T_{pid} T^T_{pid} \tag{8}$$

$$\alpha^c_{pid} \in [0,1], \kappa_{pid} \in [0,1], \tag{9}$$

$$\gamma_{pid} \in [0,1], \xi \in \mathbb{R}^+, Flag_{Train} \in \{0,1\} \tag{10}$$

In line 2, A_{tar} is a user-defined model accuracy constraint and β^c_{pid} denotes the percentage of requests leaving the internal classifier c in partition pid. In line 3, α'^c_{pid} is the profiled mean confidence during inference for the internal classifier c in partition pid, α^c_{pid} is the confidence threshold for the internal classifier c in partition pid, and β_{pid} indicates the percentage of requests leaving partition pid during inference. Notice that we introduce a flag parameter $Flag_{Train} \in \{0,1\}$ to specify whether the formulation is for training or inference.

Notation	Definition
$\pi^{Lat}_{pid}(.)$	Adapt partition F_{pid} to minimize inference latency
α^c_{pid}	Confidence thresholds for classifier c in partition pid
κ_{pid}	Output compression rate of model knowledge distillation
γ_{pid}	Output compression rate of compression layers(encoder&decoder)
γ_0	Source data compression rate of compression layers(encoder&decoder)
β^c_{pid}	Percentage of request exit at classifier c in partition pid
β_{pid}	Percentage of request exit in partition pid
x_0	Source data
Q_{pid}	Quantity of classifier in partition pid
$A^c_{\pi_{pid}}$	Observed model accuracy after adaptation
A_{tar}	User-defined model target accuracy
T^T_{pid}	Estimated transmission time
T^C_{pid}	Estimated computation time
$FLOPs^j_i(.)$	FLOPs from layer i to layer j inclusive

Table 6: Latency Optimization Formulation Notations

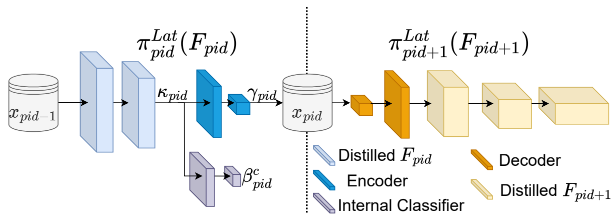

Figure 2: Illustration of latency optimization problem.

When $Flag_{Train} = 1$, the formulation optimizes for inference latency, as it considers the portion of requests that leave the internal classifiers at earlier partitions. For $Flag_{Train} = 0$, the formulation optimizes the training latency, as it ignores early exits and ensures that all internal classifiers make predictions and deep partitions are trained.

In line 4, we define the output of partition pid as $\pi_{pid}(F_{pid})(x_{pid-1})$, where the model partition F_{pid} adapted with $\pi_{pid}(.)$ takes x_{pid-1} as input. Notice that the model adaptations include the introduction of internal classifier(s) and transmission and model compression. To quantify the effect of those adaptations in training and inference latency, in line 5, we estimate the transmission delay from the adapted partition pid to $pid+1$ based on input size $Size(x_{pid-1})$, the early exiting ratio β_{pid} and the two compression ratios (γ_{pid} and κ_{pid}), for which we will discuss the specific model adaptation methods in Sec. 3.1. Then, in line 6, we estimate the transmission time for the source data to the location of the first NN partition. γ_0 is the compression ratio of the source data.

In line 7, $FLOPs^{pid}_{pid}(x_{pid-1}, \pi_{pid})$ is the profiled count of

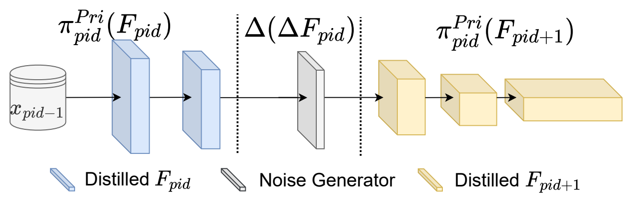

Figure 3: Illustration of Privacy optimization problem.

Notation	Definition
$\pi^{pri}_{pid}(.)$	Fine-tune partition F_{pid} for better privacy guarantee
$\tau(.)$	Noise generator method
λ	Output bounding parameter
ΔF_{pid}	Sensitivity of partition *pid*

Table 7: Privacy Optimization Formulation Notations

FLOPs (FLoating-point OPerations) for partition *pid*, given input (x_{pid-1}) and adaptation (π_{pid}). The subscript and superscript of $FLOPs^{pid}_{pid}(.)$ indicates the start and end partitions to count FLOPs. When we focus on one partition, the subscript and superscript are the same.

2.2.2 Privacy(L_p)

We study the privacy of source data in the distributed DNN training and inference application. This section explores remedies for data leakage when hidden variables are exposed and vulnerable to model inversion attacks [41, 156, 169].

Regularization [55, 89, 150, 179] methods adapt model training to resist source data reconstruction. *Perturbation* [55, 112] methods ($\tau(.)$) inject noises based on partition sensitivity (ΔF_{pid}), which gauges the range of partition output enforced by an output bounding parameter λ. We show their application in Fig. 3. There are three stages, divided by the dashed lines. The left and right portions represent model partitions *pid* and $pid+1$ enhanced by privacy-aware regularization (π^{pri}_{pid}). The middle portion represents the added perturbation $\tau(\Delta F_{pid})$. The constrained optimization problem to optimize $\pi^{pri}_{pid}, \Delta, \lambda$ is formulated below, with notations in Table 7.

$$\mathcal{L}^P = \min_{\pi^{pri}_{pid},\Delta,\lambda} \left(w_{CE}CE(\hat{y},y) - \sum_{pid=1}^{M} w_p MSE(F^{-1}_{pid}(x_{pid}), x_{pid-1})\right) \tag{1}$$

$$s.t.\ CE(\hat{y},y) \geq Thr_{CE} \tag{2}$$

$$x_{pid} = \lambda\pi^{pri}_{pid}(F_{pid})(x_{pid-1}) + \tau(\Delta F_{pid}) \tag{3}$$

$$\forall pid > 1 \tag{4}$$

The objective optimization function (line 1) balances source data privacy ($MSE(F^{-1}_{pid}(x_{pid}), x_{pid-1})$), where F^{-1}_{pid} is the inverse approximation of NN partition F_{pid}, and Cross-Entropy (CE) between the prediction $\hat{y}$ and the ground truth y subject to the CE threshold (Thr_{CE} in line 2). In line 3, we specify the forward propagation step for each fine-tuned NN partition, incorporating output bounding (λ) and noise ($\Delta(\Delta F_{pid})$) to the intermediate data.

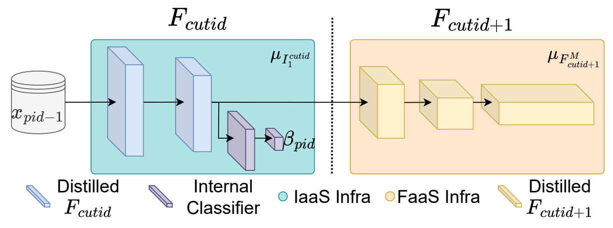

Figure 4: Illustration of cost($) optimization problem.

Notation	Definition
Latency	Latency bound
T_I	Observed Mean IaaS Time
T_F	Observed Mean FaaS Time
T^{cutid}_{cold}	FaaS function cold start time
T^{cutid}_{trans}	Transmission delay from IaaS to FaaS
$C_I(.)$	Unit cost of IaaS given VM capacity
$C_F(.)$	Unit cost of FaaS given function capacity
μ_I	VM capacity
μ_F	Function capacity

Table 8: Cost($) Optimization Formulation Notations

2.2.3 Monetary Cost(L_c)

Resource provisioning approaches based on resource cost ($) for DL training and inference tasks remain underexplored. Using detailed cost models of different cloud and edge services, an MLaaS broker can determine cost-efficient resource provisioning strategies for different workloads. In particular, for decomposable ML models, fine-grained resource provisioning and load balancing for submodels are essential to achieve cost-efficient ML training and inference.

In our formulation, we minimize the costs of provisioning cloud and edge resources by using Infrastructure-as-a-Service (IaaS) and Function-as-a-Service (FaaS) platforms at the edge and cloud. FaaS provisions serverless functions to serve user workload, which has a finer granularity cost model than IaaS. FaaS only charges users when the deployed model is used, which is suitable for *low-rate* workloads compared to IaaS resources, for example, a temporary spike in DNN inference workload [65, 69]. On the other hand, the IaaS platform automatically scales virtual machines (VMs), which have a longer cold start time needed for hardware and operating system provisioning compared to FaaS. Due to the coarse granularity of VM scaling (extra cold start time), a user tends to over-provision VMs to satisfy a service-level objective (SLO) of a dynamic ML inference workload, which wastes monetary cost of resources. Nevertheless, when the workload allows the user to fully utilize the VMs, the monetary cost of using VMs would be lower than the cost of using serverless functions. Such workloads are often characterized by a *steady high-rate* of data for processing, for example, the steady portion of a DNN inference workload or ML training tasks [72].

Motivated by recent progress in sparse model design, where only a subset of neurons are activated during training and inference [59], and the need to partition a large model to fit in edge devices [75], we model a dependent acyclic graph of submodels representing a partitioned sequential NN with internal classifiers at each partition, as shown in Fig. 4. For inference tasks, the partition F_{pid} has β_{pid} percentage of requests that exit at the internal classifier. And the partition F_{pid+1} processes the remaining portion of the requests. Notice that the shallow partitions experience steady high-rate traffic, while the deep partitions experience low-rate traffic. Thus, considering a consistent N requests per second, we can use VMs for the r_{max} portion of N requests processed by shallow partitions that can fully utilize the VMs. For all remaining requests, we use FaaS to avoid under-utilization of any individual VM.

$$\mathcal{L}^C = \min_{cutid,\mu_F,\mu_I,\alpha^k_{pid}} C_I(\mu_I)T_I \sum_{pid=1}^{cutid} \beta_{pid} + C_F(\mu_F)T_F \sum_{pid=cutid+1}^{M} \beta_{pid} \tag{1}$$

$$s.t. \quad Latency \geq T_I + T_F \tag{2}$$

$$\sum_{pid=1}^{M} \sum_{c=1}^{Q_{pid}} \beta^c_{pid} A^c_{\pi_{pid}} \geq A_{tar} \tag{3}$$

$$\beta_{pid} = \sum_{c=1}^{Q_{pid}} Pr(\alpha'^c_{pid} > \alpha^c_{pid}) = Flag_{Train} \sum_{c=1}^{Q_{pid}} \beta^c_{pid} \tag{4}$$

$$T_F = \frac{FLOPs^M_{cutid+1}(x_{cutid})}{\mu_F} + T^{cutid}_{cold} \tag{5}$$

$$T_I = \frac{FLOPs^{cutid}_1(x_0)}{\mu_I} + T^{cutid}_{trans} \tag{6}$$

$$cutid \in [1,M],\ \alpha^k_{pid} \in [0,1], \tag{7}$$

$$Flag_{Train} \in \{0,1\}, \tag{8}$$

$$\mu_F \in \{\text{FaaS Capacities}\}, \tag{9}$$

$$\mu_I \in \{\text{IaaS Capacities}\} \tag{10}$$

Focus on Inference Tasks: The optimization above focuses on consistent r_{max} DNN inference requests per second that can fully utilize VMs. It identifies the resource configurations of using IaaS, FaaS or hybrid offloading of some requests from IaaS to FaaS so they complete their processing of deep layers. The optimization adjusts the FaaS configuration (μ_F), the IaaS configuration (μ_I), internal classifier thresholds (α^c_{pid}), and the model partitioning index (*cutid*, assuming two partitions in total), which are constrained by a *latency* bound in line 2.

C_I and C_F represent the cost mappings (in \$) for different resource configurations, based on the average processing times (T_I for IaaS VM's reservation time and T_F for FaaS execution time) obtained through profiling for each forward propagation. In line 4, the percentage of forward propagations exiting at a specific NN partition *pid* (β_{pid}) is shown based on confidence thresholds (α^c_{pid}). Lines 5 and 6 define the profiled mean durations for FaaS and IaaS, respectively. For any given *cutid*, we calculate the duration by dividing a tunable capacity (μ_F or μ_I) by the required FLOPs ($FLOPs^M_{cutid+1}$ signifies the number of operations from partition $cutid+1$ to M), assuming that forward propagation does not revert to IaaS after being offloaded to FaaS. This step overestimates the utilization of each service because requests can exit the internal classifiers before reaching partition M. To estimate the expected inference duration, we multiply the empirical early exit rates ($\sum^M_{pid=cutid+1}\beta_{pid}$ for T_F and $\sum^{cutid}_{pid=1}\beta_{pid}$ for T_I, respectively, in line 1). Furthermore, we incorporate the delay for transmitting hidden variables from a VM to a serverless instance (T^{cutid}_{trans}) in T_I (line 6), because the serverless function is not yet invoked and would not incur monetary cost for FaaS (unlike a VM that keeps running). On the other hand, we also include the short cold start time of serverless function instances (T^{cutid}_{cold}) in T_F (line 5), as such cold start duration for a serverless function involves setting up hardware and loading of the model and would incur monetary cost.[2]

Next, using the resource configuration for r_{max} requests per second, we can optimize a load balancing pipeline for r_{max} requests per second to be served by the shallow layers on fully-utilized VMs. The remaining requests (including those that need further processing by deep layers) are directed to FaaS.

Adaptation to Training Tasks: Our monetary cost formulation also applies to ML training setting. Although FaaS is generally unfavorable for training tasks, we can generalize T_F, μ_F, C_F and r_{max} as duration, computation capacity, unit cost of edge resources, and training batch size, respectively. And we should set $Flag_{Train} = 0$, so that $\beta_{pid} = 0$.

Previous works have proposed various distributed deep learning training paradigms that leverage resource-specific cost models across edge and cloud environments. One example is Federated Learning (FL), where multiple nodes train local models independently and aggregate their outputs. A more recent variant, Split Federated Learning, enhances FL

[2]Notice that major FaaS providers would keep the serverless instance running after a request finishes to minimize this cold start duration [23, 45, 109, 137]. Thus, the cold start duration (T^{cutid}_{cold}) does not apply to every request, and thus the formulation overestimates the FaaS cost. Also, the cold start time of a VM is not shown in the formulation, because this formulation assumes long-running VMs serving a consistent r_{max} workload.

by offloading portions of each node's model from the cloud to the edge, thereby improving privacy and training efficiency.

These paradigms introduce new challenges, concerning transmission efficiency between the cloud and edge, as well as privacy preservation given varying depths of the edge NN partition. For example, placing a deeper NN partition at the edge enhances data privacy by limiting raw data exposure. However, this approach may increase processing delays due to the limited computational resources of edge devices. Additionally, transmission latency can be affected by the size of intermediate data outputs, which varies with different partitioning strategies. Addressing these issues requires revisiting the monetary cost models for edge and cloud resources, balancing factors such as latency, energy consumption, and privacy requirements.

3 Problem Solutions

In the preceding section, we formulated an optimization problem for deploying a dependent acyclic graph of submodels, accounting for latency, source data privacy, and resource cost ($) savings. In this section, we discuss the solutions to these optimization problems, such as early exits, compression, and privacy-preserving training and inference techniques. Furthermore, we identify open issues, including latency-sensitive selection of hidden variables, prediction of privacy level, and cost-aware dynamic NN partitioning. These techniques can serve as valuable control mechanisms for ML service providers, improving Quality of Service (QoS), and increasing revenue.

3.1 Latency

The end-to-end latency of a neural network model comprises both processing and transmission delays. Building on earlier discussions, existing work dynamically minimizes the transmission of excessive hidden variables and combines capable cloud services. This section begins by exploring dynamic deep neural network offloading [75, 167]. Then, we discuss internal classifiers which allow early exit and save computation for deep layers [37, 81, 85–87]. Furthermore, we examine transmission data and model compression approaches [39, 61, 81, 104, 117, 138, 176, 181].

3.1.1 Dynamic Partitioning

When dynamically partitioning a neural network, the computation ($FLOPs_i^j(.)$) and activation size ($Size(.)$) can be estimated based on the model weights and input size [10, 144]. Thus, delays, especially inference durations (T_{pid}^T and T_{pid}^C) can be modeled using regression methods, by profiling the NN model, across the cloud and resource-limited edge environments [75, 167]. Previous work estimates transmission and processing delays for various model configurations, factoring

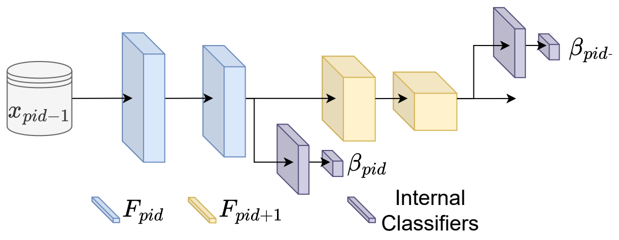

Figure 5: Internal Classifier Architecture: Each internal classifier allows requests to exit in the middle of an NN. For example, β_{pid} of requests exit at NN partition F_{pid}.

in computation resources and input sizes to devise a deployment plan for M NN partitions that minimizes latency and energy consumption [75]. However, feasible solutions may not always exist for a given model architecture or environment, for example, when resource availability is constrained. Next, we explore orthogonal methods to reduce demands on transmission and processing resources.

3.1.2 Early Exits

Background. Shallower layers of a DNN model extract high-level features which can be sufficient for accurate request classification, while deeper layers can focus on certain fine details, sometimes resulting in misclassification. The issue of *overthinking* was diagnosed in the ShallowDeepNet paper [76]. Related research [76, 163, 183] addresses this concern, proposing the reuse of features extracted from various layers for prediction to improve convergence time and inference cost via *internal classifiers* (Fig. 5).

Internal classifiers were first proposed for cost-efficient DNN inference in BranchyNet [146]. These classifiers share a structure similar to traditional NN classifier layers, typically comprising feature reduction (pooling) layers, fully connected layers, and a softmax activation function. However, internal classifiers are attached to the hidden layers. To trigger early exits, one can configure a threshold [76] for the Bayesian probabilities of class predictions at each classifier [50, 129].

In deep learning (DL) tasks, incorporating early exits and residual connections at various internal layers of a DL model allows for better utilization of insights during training and inference. This integration improves prediction accuracy while minimizing computational waste. Early exits prevent excessive forwarding of requests (hidden variables) to deep layers for classification. As exemplified in Fig. 5, an internal classifier allows β_{pid} portion of requests to exit the model partition F_{pid}, highlighted in blue. Subsequent works have leveraged this approach to minimize inference delay in distributed inference applications.

Methods. Recent research [61, 81, 147] models the relationship between the confidence threshold (α) of internal classifiers and the proportion of early exits (β) when formulating delays. As shown in our latency optimization framework, this allows tuning β through α to meet a specific mean latency target (lines 2 and 3.) However, lowering the confidence threshold can negatively impact model accuracy, as it allows requests to exit with reduced output probabilities, potentially resulting in decreased accuracy.

When accuracy is low, previous studies explore combining internal classifiers with dynamic layers offloading to the edge or cloud based on network conditions, as in lines 4, 5, and 7. For example, SPINN [81] empirically demonstrates that under high and stable WAN bandwidth, more layers can be offloaded to cloud nodes. The increased computational capacity compensates for additional communication delays, reducing overall latency. In contrast, when network bandwidth is limited, the approach shifts more layers to resource-limited edge nodes. Despite an increase in processing time at the edge, overall latency is optimized by minimizing reliance on WAN communication.

3.1.3 Input and output compression

Background. When training an ML model, not all available features are necessary for a classification task. Feature engineering addresses this by combining features or eliminating unnecessary ones. Apart from traditional statistical or heuristic methods [152], Deep Neural Networks (DNNs), specifically Auto-Encoder NNs, can facilitate feature selection to preserve prediction performance [138]. An Auto-Encoder NN consists of two components: an encoder, which transforms inputs to a condensed output representation, and a decoder, responsible for inverting the dimensionality reduction [131]. On the other hand, model compression methods can also reduce feature size. These methods will be explored further in Sec. 3.1.4.

Methods. In our latency optimization ($\mathcal{L}^L$), we denote the cropping and compression of input data with rate γ_0 (line 6). The compression rate of intermediate data achieved through model compression is denoted as κ_{pid}, while the rate achieved through feature engineering methods such as Auto-Encoder is represented as γ_{pid}. We encapsulate the computation overhead of Autoencoder NN in the model transformation π_{pid}.

Heuristic-based compression methods, such as JPEG for image inputs, can help reduce feature dimension. In particular, certain activation functions, for example relu [20], produce zero or near-zero outputs, allowing compression from a dense matrix into a sparse matrix that is storage and transmission efficient [61]. Moreover, related works [61, 66, 101, 135] explore the quantization of weights and intermediate data representations. Rather than using double precision floats, these methods consider 8-bit [66] or in the extreme case single-bit [101] approximations.

Other works explore content-based transmission compression methods. For example, in an *AMBER Alert* system, if the model only requires identifying a car or person in the scene, the edge device only transmits cropped images focusing on Region of Interest (ROI) to the cloud for analysis [130]. This approach minimizes transmission delay, although accuracy may vary depending on the effectiveness of the cropping techniques. Moreover, some studies suggest that focusing on the relevant data not only reduces transmission costs, but also enhances accuracy [117].

Similarly, previous work has applied ML-based dimensionality reduction tools to reduce transmission data and maintain accuracy. One idea is to introduce a *bottleneck* between two neural network partitions using an Auto-Encoder NN [39, 63, 104, 138, 170, 181]. This Auto-Encoder is trained by minimizing the Mean Squared Error (MSE) loss between the input and output data. In this setup, the encoder maps the intermediate data into a more space-efficient latent space, effectively reducing the channels, width, and height. The decoder, which serves as an approximation of an inverted encoder function, reconstructs the input of the previous partition using the compressed intermediate data. This approach enables a compact representation of intermediate data, enhancing efficiency without significant loss of accuracy.

To better maintain the accuracy of the model, recent studies [63, 138, 170] propose optimizing the entire model, encompassing both the backbone and the Auto-Encoder NN. This approach enhances model accuracy by guiding the Auto-Encoder NN to focus on features essential for predictions. However, finding an optimal compressed feature space for both high accuracy and high γ_{pid}) remains a challenging task that requires extensive hyperparameter tuning. To address this, recent research [62] uses explainable AI techniques, including Integrated Gradients [145], to construct an intermediate data space that emphasizes features with the greatest impact on predictions.

Summary of data compression methods: In our latency optimization framework, we adjust the data compression rates γ_{pid} and κ_{pid} to minimize transmission overhead. The introduction of data compression adds FLOPs in each partition (π_{pid} in line 4), creating a trade-off among reduced transmission overhead, increased computation overhead, and potentially compromised model accuracy. Various data compression methods are detailed in Table 9, highlighting the practicality of both Auto-Encoder and quantization techniques as they are broadly model and data agnostic. However, an Auto-Encoder offers greater flexibility compared to quantization, which enables fine-tuning the compression model (encoder), inversion model (decoder), and feature ranking techniques (such as explainable AI tools) to optimize latency and accuracy based on the user's specific use case. While an Auto-Encoder introduces additional computational overhead, a quantized model and activations also require specialized training tools due to the discrete space. For example, stochastic gradient descent (SGD) must be adapted to accommodate the discrete space [66, 114].

	Pre-Processing [117]	Heuristic [61]	Quantization [61, 66, 101, 135]	AE [39, 63, 104, 138, 170, 181]	AE(XAI) [62, 63, 104, 138, 170]
Computation	low	low	low	medium	medium
Compression	medium	medium	medium	low	low
Accuracy	high	high	medium	medium	high
Practicality	low	low	high	high	medium

Table 9: Comparison of Input and Intermediate Data Compression Methods: The accuracy of the pre-processing method [117] depends on the ability of the algorithm to accurately identify and crop the features of interest before sending data to the model (low practicality, high accuracy given good cropping algorithm, low extra computation for cropping input, and overall medium compression rate for cropping). Heuristic-based compression algorithms, like clustering for zeros, rely on user expertise (low practicality, high accuracy, low extra computation, and overall medium compression rate depending on the inputs and heuristics applied). Intermediate data quantization shortens data representation but may impact accuracy (high practicality, medium accuracy, and medium compression rate compared to other task-oriented methods) and demands an adapted optimization method for discrete space (low extra computation). The Auto-Encoder (AE) can be applied to various data representations (high practicality) and can be adapted to different ML tasks by using shallower layers to minimize computation overhead (medium computation overhead) or designing smaller latent spaces to create a narrow bottleneck that tradeoffs accuracy (medium accuracy and low compression rate). Naïve input or intermediate data compression can significantly compromise model accuracy if the features selected for transmission are suboptimal. In contrast, AE approaches leveraging explainable AI (XAI) tools selectively transmit crucial features for classifications, reducing transmission delay while maintaining high accuracy (overall medium practicality based on feature selection methods, overall medium computation demand with AE, high model accuracy, and low compression rate).

3.1.4 Model Compression and Knowledge Distillation

Background. Deep Learning (DL) models can be customized to meet specific Machine Learning (ML) tasks and computational constraints. For latency-sensitive applications, simplifying the model can enable faster response times. Such simplification can be achieved through quantization [91], layer skipping [96], adding, removing, or editing the layer blocks of a neural network [168], and knowledge distillation [56].

A common tradeoff of reducing model parameters is potential accuracy degradation. However, such a drawback might be tolerable given the use case or in certain settings the model would not suffer a significant accuracy drop, as model simplification can be considered a form of regularization that mitigates overfitting and discourages shortcut learning [43, 67]. Thus, model simplification is considered a versatile approach applicable across many different ML systems, achieving short processing times without significant accuracy loss.

For example, in large ML systems, such as large language models (LLM), the Mixture of Experts model architecture [48, 139] decomposes a high-dimensional model into smaller *experts* with a router NN selecting a subset of submodels for each request, reducing computational demands. In certain LLM-based chatbot applications, chats generated by simplified LLMs on user devices can be enhanced by constraining the output space or leveraging cached outputs from full-sized LLMs deployed in the cloud. This allows the final chatbot responses to match full-sized LLM quality while maintaining high throughput on the device [7, 34].

On the other hand, distilled or quantized LLMs can assist original LLMs in speeding up chat generation. Traditionally, chat generation proceeds sequentially token-by-token, resulting in low throughput. Instead, a smaller LLM can speculatively generate the next t tokens, which the original LLM then verifies and either accepts or rejects them in parallel based on the speculatively generated context. This *speculative decoding* approach significantly boosts chatbot throughput [27, 84, 108].

We first discuss simplifications for standalone ML systems and then generalize them to the distributed setting.

Quantization or compression of neural network (NN) weights and activations is a popular technique to reduce computation complexity [91]. Previous work introduces Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). For small models or large models with aggressive quantization, recent studies have shown that using lower precision during training, for example 8 bits [66] and 1.58 bits [102], for NN weights can achieve accuracy comparable to higher precision representations. In contrast, for very large models including LLMs, quantization training introduces significant overhead. In AWQ [92], SmoothQuant [164] and OPTQ [40], researchers adjust the *Scale* and *Zero Point* (origin) for weights in pretrained models with static analysis of activation and weights. Quantization improves inference latency and reduces storage requirements for various deep learning (DL) inference tasks. For model compression, in EIE [52], the authors introduce a novel representation and matrix multiplication algorithm that omits most common values in activations, optimizing computation and storage efficiency.

We can also dynamically skip layers or make predictions before the neural network (NN) model completes its full pass without modifying the base model. This approach, known as *dynamic inference* [53, 166], involves training additional gating networks to determine which layers or channels within a layer to skip, indicated by 0 (skip) or 1 (use). In DDI [159] and SkipNet [157], the authors use a Long Short-Term Memory (LSTM) NN to make skipping decisions.

Methods focusing on weight and layer changes can be sub-optimal in reducing latency, as they introduce minimal structural changes to the model. Knowledge Distillation [47, 56] trains a lightweight high-performance model (Student Model) using the inputs and outputs (including hidden variables or logits) of a more complex model (Teacher Model). As the Teacher Model is often over-parameterized, the Student Model can attain similar accuracy with reduced computational complexity. Furthermore, the Teacher Model can also be further refined with weighted outputs from the Student Models (*soft labels*) and ground truth labels in a *student-student* knowledge distillation setting [174, 176, 180]. In this process, the output of the student model serves as a form of regularization to prevent overfitting.

Methods. In our latency optimization formulation in a distributed setting (Sec. 2.2.1), we use π_{pid} to represent all adaptations, including model compression. The function $FLOPs_i^j(.)$ estimates the number of floating point operations given different partition configurations π_{pid} and partition input size $Size(x_{pid-1})$. In a dense NN, the FLOP count is linearly proportional to the number of model weights and the size of input data, so more layers or larger input sizes lead to longer processing times. To minimize computation latency, we can employ model compression or knowledge distillation methods for each NN partition. As a beneficial byproduct, model compression can also reduce the size of hidden variables which improves κ_{pid} introduced in Sec. 3.1.3.

Quantization [63, 88, 114] and neuron skipping [61, 83, 181] can be applied to each NN partition. Intuitively, weights close to zero contribute little to classification and can be pruned to save processing time without significantly affecting accuracy. In CLIO [61], a certain percentage of weights, sorted by distance to zero, is ignored. However, with a higher compression rate, such approaches suffer from low prediction performance.

To maintain accuracy in cases of significant compression for resource-constrained edge partition NNs, Lee et al. [83] suggest the use of a deep decoder NN at the cloud node. The deep decoder with high inversion approximation capabilities compensates for the aggressively compressed NN partition, helping to preserve the model's accuracy.

Instead of compressing a base model, training a lightweight model replacement can effectively scale model size down to fit the capacity constraints. With limited edge capacity, applying knowledge distillation to partitions could save processing time with minimal loss in accuracy [104].

Summary of model compression methods: In our latency optimization framework (Sec. 2.2.1), π_{pid} represents model adaptations including both model compression and distillation, which reduce model complexity yielding a placement with minimal processing time. This section reviewed the application of quantization, knowledge distillation, and weight pruning, each with strengths and weaknesses. While these methods are orthogonal and should be evaluated together to optimize model complexity and placement, they vary in terms of practicality and computational overhead. Table 10 presents a summary and comparison of these model compression techniques.

Among these techniques, knowledge distillation is the most configurable, offering various student model designs and distillation approaches to achieve high model accuracy. Thus, it is considered the most practical, but with the highest computation overhead. In contrast, quantization is less configurable, so we position it at medium practicality but with the lowest computation overhead. The effectiveness of weight-pruning depends largely on the underlying data distribution. For example, pruning weights close to zero is a well-explored method to maintain accuracy while reducing computational complexity. However, mask-based pruning methods may require training specific to each source data distribution [33], leading to medium practicality but low computation overhead for each mask.

3.1.5 Open Issue: Dynamic Feature Extraction Tuning

In sequential DNN models, each layer extracts different features and feeds the hidden variables to the subsequent layers. Previous works have explored combining knowledge extracted at different layers. Residual NNs [54] propagate primitive features to the deeper layers. Early exit architectures [146] leverage low-level features for classification. Zero Time Waste [163] combines features at different levels to improve classification. Such works enable dynamic DNN offloading by incorporating classifiers at partitions of the model. However, when model partitions are being offloaded to a user device for low transmission latency, to the cloud for low processing latency, or to the edge for balanced delays, the inference or training service should remain active. The additional workload introduced to the user device, edge or cloud due to offloading should not degrade service performance.

Understanding the reasons for offloading helps optimize performance. Offloading partitions to the user device typically addresses network transmission bottlenecks, while shifting them to the cloud helps alleviate computational limitations at the edge. To ensure consistent performance, particularly for critical tasks identified by the application, we discuss a potential approach that focuses on the initial partitions at the edge. The core idea is to ensure that shallow DNN partitions at the edge achieve high accuracy specifically for these critical tasks. This prioritization might come at the expense of reduced accuracy for non-critical tasks, but allows the system to rely on efficient, local edge computations for its most important functions, preserving their performance and minimizing

	Quantization [63, 88]	KD [104]	Weight-Pruning [61, 83, 181]
Computation	low	high	low
Accuracy	medium	high	high
Practicality	medium	high	medium

Table 10: Comparing Model Compression and Knowledge Distillation (KD) Methods: KD preserves essential weights to ensure high model accuracy, which involves model building and training (high computation). However, it can be applied to models of any size (high practicality). Quantization reduces the precision of all weights. While it is generally task-agnostic, model accuracy can degrade (medium accuracy and practicality). The process quantizes the representation and adapts the optimization method, which is lightweight (low computation). The effectiveness of weight pruning depends on the distribution of weights and the specific task (medium practicality, high accuracy, and medium weight size). Heuristic-based pruning method also has low computation complexity.

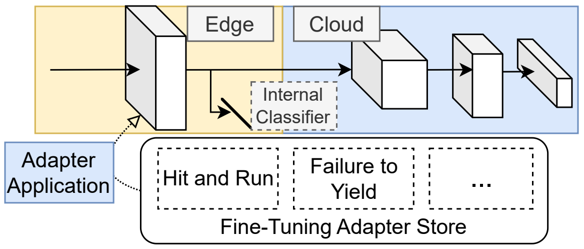

Figure 6: Dynamic feature extraction tuning: Shallow edge partitions apply task-specific adapters (e.g., LoRA [58]) to preserve recall for critical tasks, enabling early exits and avoiding deeper cloud inference during model offloading.

disruptions when deeper partitions are dynamically relocated.

One way to implement this strategy for inference tasks involves adapting the feature extraction process in the shallow partitions at the edge. By tailoring these shallow partitions to focus specifically on features relevant to critical tasks, shallow internal classifiers integrated at these early partitions can achieve high confidence predictions for those specific tasks. This enables critical requests to exit early with low latency directly from the edge. However, this specialization may lead to higher false positive or false negative rates for non-critical task handled by these adapted shallow layers.

Since critical tasks represent a subset of the full task space addressed by the complete model, we can use the outputs of a model partition that performs well on these critical tasks to guide and train the shallow partitions deployed at the edge. This form of feature adaptation is conceptually similar to knowledge distillation, particularly self-distillation [176]. To implement this efficiently, Parameter-Efficient Fine-Tuning (PEFT) methods, such as LoRA (Low-Rank Adaptation) [58], can be employed. As illustrated in the system design (Fig. 6), an offline process can fine-tune and generate lightweight LoRA adapters tailored for specific subtasks (e.g., different types of traffic violations) and save them in a *Fine-Tuning Adapter Store*. Each compact adapter can be associated with an embedding fingerprint, allowing for rapid retrieval and deployment at the edge when needed.

Applying this concept to traffic surveillance, for instance, involves attaching these lightweight LoRA adapters to the shallow DNN partitions deployed at the edge. These adapters enhance the shallow partitions' ability to accurately classify specific life-threatening traffic violations during periods of model partition offloading. Consequently, the system can maintain high recall for these critical violations by primarily utilizing the computationally inexpensive shallow edge layers, ensuring timely detection even when deeper network segments are being offloaded.

3.2 Privacy

Distributed DL systems processing sensitive personal data raise data leakage concerns. Private data should be inaccessible outside the customer's infrastructure or protected from reconstruction during transmission over wide area networks (WAN). As described in Section 2.1.2, an adversary could reconstruct the intermediate data transmitted between DNN partitions [150] using an Auto-Encoder Neural Network. To protect against this vulnerability for model training and inference, previous research adds *Perturbation* to intermediate data [55, 112] or incorporates a *Regularization* step during training [55, 89, 150, 179]. Such methods preserve only essential features for ML tasks and remove sensitive information.

3.2.1 Perturbation

Background. Differential privacy (DP) has been used to improve privacy in statistical databases by adding noise to query outputs proportional to the *sensitivity* of the query [36, 175]. Consider a query $f : D \to R$ on a dataset D with samples $x, x^{'} \in D$. The global sensitivity Δf of this query is defined as:

$$\Delta f = \max_{x,x^{'}} \|f(x) - f(x^{'})\| \quad (1)$$

Users can set the *Privacy Budget* ε. Then, noise can be drawn from a Laplace distribution, $X \sim Laplace(\frac{\Delta f}{\varepsilon})$, to achieve the desired level of privacy based on various privacy definitions.

More specifically, the probability density function (PDF) is $p(x) = \frac{1}{2b}e^{\frac{-\|x\|}{b}}$, where $b = \frac{\Delta f}{\varepsilon}$. The value of ε can be determined by a grid search against the attack model. With a smaller ε, we spread the PDF and introduce more diverse noise to the output, so less information is preserved.

In practice, finding the global sensitivity Δf is challenging as it requires testing all inputs. Instead, previous work bounds the sensitivity in model partition output [18, 175].

$$x'_{pid} = \frac{x_{pid}}{max(1, \frac{\|x_{pid}\|}{C})} \quad (1)$$

where C is the clipping threshold. In this way, $\|x'_{pid}\| < C$. Notice that clipping modifies the hidden variables which leads to accuracy degradation. To optimize C, the common practice is to set the median of x_{pid} based on the training dataset [18].

Previous studies apply this practical DP implementation in DP-SGD [18, 173] during training to mitigate the risk of reconstructing training datasets from trained models. They add Gaussian noise to gradients, reducing the model's sensitivity to individual training samples. As a result, the distribution of prediction confidences for training dataset samples is similar to other samples, preventing over-concentration on the true label. It complicates membership inference attacks, in which adversaries deduce whether a sample was part of the training data based on prediction logits [42].

In edge inference and training settings, recent work suggests injecting noise to hidden variables that obscure sensitive information, for example, race, age, or gender, transmitted over the Internet [55, 112].

Methods. In our privacy optimization formulation (Sec. 2.2.2), in line 3, an MLaaS system can inject noise to intermediate data ($\tau(\Delta f)$) based on its sensitivity Δf. With differential privacy, recent studies [55, 103, 112, 175] have developed fitted noise layers that either sample noise from a distribution or nullify specific entries. This approach is highly flexible, allowing users to choose different noise layers to append to the final layer on the edge device when the source data distribution changes. The noise injected during training and inference complicates the inversion approximation used by the attacker. Meanwhile, the model retains its capacity to extract relevant information for accurate predictions.

3.2.2 Regularization

Background. We can also solve the privacy of the source data as an optimization problem. One approach is to incorporate source data privacy as a secondary objective by adding a regularization term to the loss function. Thus, we encourage the model to preserve only the features that contribute to prediction. On the other hand, deep edge neural networks (NNs) with non-invertible hidden variables, such as rectangular matrices, are harder to approximate an inversion matrix. Therefore, we can optimize the placement of NN partitions to maximize the privacy level of the source data.

Methods. In our privacy optimization formulation (Sec. 2.2.2), we incorporate the privacy loss, exemplified as $MSE(F^{-1}_{pid}(x_{pid}), x_{pid-1})$ into the loss function in line 1. The mean square error gauges differences between the reconstructed and original source data. Then, we can tune the privacy level of model inference by specifying hyperparameters w_{CE} and w_p for training [89, 150, 179] and model partition placements [55, 179].

There are ways to incorporate privacy objectives into model training. For example, we can include a distance correlation loss function, comparing intermediate data and source data in addition to the Cross-Entropy loss [150]. Alternatively, additional training epochs can be dedicated to optimizing the privacy objective [179]. For more task-specific solutions, ResSFL [89] introduces a privacy loss function that compares the source data and the reconstructed data derived from intermediate data using a *decoder* following the threat model in model inversion attacks. Thus, by designing decoders with different capacity, the model can defend against different model inversion attacks.

3.2.3 Open Issue: Privacy and Accuracy Estimation

By profiling the model partition running time, the size of hidden variables and gradients, and the available bandwidth, we can fit a regression model to estimate the processing and transmission delays of different offloading plans for DL models [75]. However, the relationship between privacy-preserving training approaches and their resulting privacy guarantees and impact on model accuracy is often poorly understood before training. As shown in our privacy formulation (Sec. 2.2.2, line 1), the results of $CE(\hat{y}, y)$ and $(F^{-1}_{pid}(x_{pid}), x_{pid-1})$ given different combinations of hyperparameters $\{\pi^{pri}_{pid}, \Delta, \lambda\}$ only become apparent after the model training converges.

This leads to two major challenges. First, hyperparameter tuning becomes extremely time-consuming, requiring numerous full training cycles. Second, the tuning process itself is vulnerable to privacy breaches, as exchanging hidden variables and gradients between partitions during tuning can expose data before effective privacy configurations are identified and implemented. Although some heuristics might offer limited guidance (e.g., higher compression generally correlating with lower cross-entropy), systematically narrowing the search space for optimal privacy hyperparameters in distributed training remains an underexplored and critical research area.

Some related work attempts to mitigate privacy risks during training by employing transfer learning [89]. This strategy involves first training a privacy-aware DNN model with a publicly available image classification dataset, where data leakage is not a concern. Then, this pre-trained model is fine-tuned using the private dataset via transfer learning. The intention is to provide a privacy-aware starting point, potentially reducing data leakage risks during the initial epochs of training on the sensitive data. However, this approach faces limitations, particularly when adapting from a relatively simple pre-training task (e.g., CIFAR-10) to a significantly more complex target task (e.g., CIFAR-100). In such scenarios, effective transfer learning may be infeasible, necessitating extensive fine-tuning or even complete model retraining on the private data to achieve acceptable accuracy. Consequently, again each training trial during the lengthy hyperparameter tuning phase becomes vulnerable to privacy leaks and consumes significant computational resources. Thus, efficiently and securely tuning hyperparameters for privacy-aware training remains a largely unresolved issue.

3.3 Cost($)

Deep learning tasks require substantial resources due to their high computation and memory demands. For example, the recent deep transformer models encounter memory bottlenecks when loading and saving attention layers [44]. Previous work discusses splitting model states [122], kernel fusion techniques [29], and sparse attention mechanism [172] to reduce GPU memory demands and transmission between CPU and GPU memory.

Previous works emphasize low-level DL task scheduling. Instead, this survey focuses on the higher level aspects of resource provisioning. Organizations developing intelligent applications using an MLaaS system often face budget constraints when provisioning resources from cloud providers. For instance, the daily running cost for ChatGPT can reach up to $700,000 [9]. Furthermore, cloud services have different cost models that factor in billing granularity, scaling speed, availabilities, etc. A cost-efficient MLaaS system should strategically choose, configure, and load balance the cloud resources to optimize expenses while meeting performance demands.

3.3.1 Methods

To bridge this gap and promote ML systems with low monetary cost of cloud resource usage, various organizations provide services to construct intelligent applications on diverse infrastructures, with an emphasis on minimizing costs. For example, Redhat OpenShift AI [8] provides a container-based Machine Learning as a Service for on-demand model serving.

Previous research has examined the dynamic scheduling of neural network and model partitions across edge and cloud resources [61, 81, 97, 119, 182], as well as specific cloud services, such as Function as a Service (FaaS) [69], to balance processing and transmission demands while minimizing expenses. However, detailed cost analyses using real-world cloud resources for low-cost ($) ML serving and training remain limited. Many studies model resource expenses on the edge and in the cloud using generic unit costs [97, 119, 182]. However, the specific provisioning factors for each edge and cloud resources, including container cold starts and billing time granularity, are critical to minimizing real-world cloud usage costs.

3.3.2 Open Issue: Monetary Cost Optimized Resource Provisioning for ML Inference

The unique cost models of cloud services influence ML resource provisioning decisions. In LIBRA [124], they identify a *Cost Indifference Point* showing high steady-rate traffic is best served by reserved VMs and low-rate traffic can be handled by FaaS for cost-efficiency. Serverless (FaaS) platforms provide fine-grained resource provisioning with response times measured in milliseconds, making them ideal for dynamic or transient workloads and for minimizing idle resource costs [49]. In contrast, reserved VMs, although slower to respond to QoS targets, offer a lower cost per request for sustained workloads that fully utilize the VMs [137].

To build a ML system with low monetary cost of resource usage, studies of ML model designs and cloud services provisioning systems have introduced new research problems, including cloud versus edge routing based on input embeddings [32, 136], ensemble model serving [59, 77, 120, 178], etc. We also speculate that a fine-grained resource provisioning approach for individual NN partitions can save resource monetary cost for partitioned NN with internal classifiers.

When offloading a DNN, the NN partitions comprise a Directed Acyclic Graph (DAG), where partitions extract features and pass them to subsequent layers. In partitioned neural networks with internal classifiers, requests can leave early at shallow layers, resulting in reduced workloads for deeper layers. Relying only on Virtual Machines (VMs) could lead to idle resources, as all VM instances must meet the latency requirements for every request. To optimize monetary cost, a high-granularity resource provisioning approach, such as FaaS, can be employed for deeper layers, scaling resources dynamically based on workload demands. As our initial evaluation shows, for a consistent workload with r_{max} requests per second and a partitioned NN with internal classifiers allowing most requests to exit in the middle of the NN, switching from only using a large VM (c6i.xlarge) to Hybrid Offloading hidden variables from a small VM (c6i.large) to a serverless instance (8845 MB) saves from 5% to 50% of monetary cost. These savings were achieved across evaluations with different Service Level Objectives (SLOs), assuming the conservative worst-case scenario where all requests traverse the entire network. In general, a more relaxed SLO allows running more partitions on the small VM and reserving the serverless instance mainly for infrequently used deep partitions, potentially increasing savings. Conversely, under very strict SLOs, finding a practical Hybrid Offloading or FaaS configuration may be difficult, an area we plan to investigate further in future work.

4 Conclusion

In this survey, we develop a systematic evaluation of state-of-the-art model offloading methods and model adaptations. Previous work focused on offloading full models between the resource-constrained edge and the core cloud, or the user devices and the edge servers. However, given the popularity of IoT and mobile computing devices, DL services with partitioned NN models at the edge have gained more attention due to privacy concerns and low latency requirements. Since the MLaaS broker model preferred by smaller organizations is based on cloud services [19,22,161], there is a huge potential for future edge-cloud-fused MLaaS systems to better serve the growing demands of edge intelligence applications.

We have discussed common issues for edge intelligence applications, including transmission delay, processing delay, and privacy guarantees. Then we further studied the corresponding solutions as control knobs for an MLaaS broker to maintain QoS in resource-constrained edge or client devices.

We also suggest future directions to reduce resource cost ($) and dynamically adapt offloading decisions based on network conditions and source data distribution. In general, there are many open issues and ongoing research in edge intelligence. We believe that this survey serves as a starting point for future advancements in this area.

References

[1] Amazon ECS. https://aws.amazon.com/ecs/. Access Date: 2022-11-04.

[2] Amazon Lambda. https://aws.amazon.com/lambda/. Access Date: 2022-11-04.

[3] Amazon Lambda Edge. https://aws.amazon.com/lambda/edge/. Access Date: 2022-11-04.

[4] AWS IoT Greengrass. https://aws.amazon.com/greengrass/. Access Date: 2022-11-04.

[5] AWS Local Zones. https://aws.amazon.com/about-aws/global-infrastructure/localzones/. Access Date: 2022-11-04.

[6] AWS Wavelength. https://aws.amazon.com/wavelength/. Access Date: 2022-11-04.

[7] Guidance. https://github.com/guidance-ai/guidance. Access Date: 2024-08-28.

[8] Red Hat OpenShift AI Accelerates Generative AI Adoption Across the Hybrid Cloud. https://www.redhat.com/en/about/press-releases/red-hat-openshift-ai-accelerates-generative-ai-adoption-across-hybrid-cloud. Access Date: 2023-12-14.

[9] The Inference Cost Of Search Disruption – Large Language Model Cost Analysis. https://www.semianalysis.com/p/the-inference-cost-of-search-disruption. Access Date: 2023-12-14.

[10] How to Measure FLOP/s for Neural Networks Empirically? https://www.lesswrong.com/posts/jJApGWG95495pYM7C/how-to-measure-flop-s-for-neural-networks-empirically, 2021. Access Date: 2023-12-16.

[11] Xiph.org Video Test Media [derf's collection]. https://media.xiph.org/video/derf/, 2021. Access Date: 2023-12-16.

[12] Amazon Rekognition. https://docs.aws.amazon.com/rekognition/index.html, 2022. Access Date: 2022-11-04.

[13] Amazon SageMaker. https://aws.amazon.com/sagemaker/, 2022. Access Date: 2022-11-04.

[14] Amazon SageMaker Edge. https://aws.amazon.com/sagemaker/edge/, 2022. Access Date: 2022-11-04.

[15] Amazon SageMaker NEO. https://aws.amazon.com/sagemaker/neo/, 2022. Access Date: 2022-11-04.

[16] Bring Insights and Data Closer to Customers with Edge Computing. Technical report, Redhat, February 2022. URL: https://www.redhat.com/rhdc/managed-files/cl-bring-insight-data-customer-edge-computing-whitepaper-f30856pr-202202-en.pdf.

[17] Intro to Autoencoders. https://www.tensorflow.org/tutorials/generative/autoencoder, 2022. Access Date: 2022-11-04.

[18] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 308–318, New York, NY, USA, 2016. Association for Computing Machinery. https://doi.org/10.1145/2976749.2978318.

[19] Adobe. Commerce Cloud Infrastrcture Overview, 2023. URL: https://experienceleague.adobe.com/en/docs/commerce-operations/implementation-playbook/infrastructure/cloud/overview.

[20] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU), 2019. arXiv:1803.08375.

[21] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. CarMap: Fast 3D Feature Map Updates for Automobiles. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, NSDI'20, page 1063–1082, USA, 2020. USENIX Association.

[22] Amazon. AWS and Adobe, 2024. URL: https://aws.amazon.com/partners/adobe/.

[23] Amazon Web Services. Lambda Runtime Environment: Cold Start Latency. https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtime-environment.html#cold-start-latency, 2025. Accessed: 2025-05-01.

[24] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V. Krishnamurthy, and Amit K. Roy-Chowdhury. Frugal Following: Power Thrifty Object Detection and Tracking for Mobile Augmented Reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, SenSys '19, page 96–109, New York, NY, USA, 2019. Association for Computing Machinery. https://doi.org/10.1145/3356250.3360044.

[25] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection, 2020. arXiv:2004.10934.

[26] Zhuoqing Chang, Shubo Liu, Xingxing Xiong, Zhaohui Cai, and Guoqing Tu. A Survey of Recent Advances in Edge-Computing-Powered Artificial Intelligence of Things. *IEEE Internet of Things Journal*, 8(18):13849–13875, 2021. https://doi.org/10.1109/JIOT.2021.3088875.

[27] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating Large Language Model Decoding with Speculative Sampling, 2023. URL: https://arxiv.org/abs/2302.01318, arXiv:2302.01318.

[28] Kaifei Chen, Tong Li, Hyung-Sin Kim, David E. Culler, and Randy H. Katz. MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, page 292–304, New York, NY, USA, 2018. Association for Computing Machinery. https://doi.org/10.1145/3274783.3274834.

[29] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL: https://openreview.net/forum?id=H4DqfPSibmx.

[30] Shuiguang Deng, Hailiang Zhao, Binbin Huang, Cheng Zhang, Feiyi Chen, Yinuo Deng, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Cloud-Native Computing: A Survey From the Perspective of Services. *Proceedings of the IEEE*, 2024.

[31] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL: https://openreview.net/forum?id=OUIFPHEgJU.

[32] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing. In *The Twelfth International Conference on Learning Representations*, 2024. URL: https://openreview.net/forum?id=02f3mUtqnM.

[33] S. Ding, L. Zhang, M. Pan, and X. Yuan. PATROL: Privacy-Oriented Pruning for Collaborative Inference Against Model Inversion Attacks. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 4704–4713, Los Alamitos, CA, USA, jan 2024. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/WACV57701.2024.00465, https://doi.org/10.1109/WACV57701.2024.00465.

[34] Yucheng Ding, Chaoyue Niu, Fan Wu, Shaojie Tang, Chengfei Lyu, and Guihai Chen. Enhancing On-Device LLM Inference with Historical Cloud-Based LLM Interactions. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 597–608, New York, NY, USA, 2024. Association for Computing Machinery. URL: https://doi-org.ezproxy.bu.edu/10.1145/3637528.3671679, https://doi.org/10.1145/3637528.3671679.

[35] Sijing Duan, Dan Wang, Ju Ren, Feng Lyu, Ye Zhang, Huaqing Wu, and Xuemin Shen. Distributed Artificial Intelligence Empowered by End-Edge-Cloud Computing: A Survey. *IEEE Communications Surveys & Tutorials*, 25(1):591–624, 2023. https://doi.org/10.1109/COMST.2022.3218527.

[36] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[37] Maryam Ebrahimi, Alexandre da Silva Veith, Moshe Gabel, and Eyal de Lara. Combining DNN Partitioning and Early Exit. In *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '22, page 25–30, New York, NY, USA, 2022. Association for Computing Machinery. https://doi.org/10.1145/3517206.3526270.

[38] Melike Erol-Kantarci and Sukhmani Sukhmani. Caching and Computing at the Edge for Mobile Augmented Reality and Virtual Reality (AR/VR) in 5G. In Yifeng Zhou and Thomas Kunz, editors, *Ad Hoc Networks*, pages 169–177, Cham, 2018. Springer International Publishing.

[39] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. BottleNet: A Deep Learning Architecture for Intelligent Mobile Cloud Computing Services. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019. https://doi.org/10.1109/ISLPED.2019.8824955.

[40] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL: https://openreview.net/forum?id=tcbBPnfwxS.

[41] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery. https://doi.org/10.1145/2810103.2813677.

[42] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery. https://doi.org/10.1145/2810103.2813677.

[43] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut Learning in Deep Neural Networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

[44] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. AI and memory wall. *IEEE Micro*, 2024.

[45] Google Cloud. Cloud Functions Execution Environment: Instance Lifespan. https://cloud.google.com/functions/docs/concepts/execution-environment#instance-lifespan, 2025. Accessed: 2025-05-01.

[46] Google Workspace Admin Help. Generative ai in google workspace privacy hub. https://support.google.com/a/answer/15706919?hl=en, 2024. Accessed: 2025-04-20.

[47] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge Distillation: A Survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

[48] Sam Gross, Marc'Aurelio Ranzato, and Arthur Szlam. Hard Mixtures of Experts for Large Scale Weakly Supervised Vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6865–6873, 2017.

[49] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Mahmut Taylan Kandemir, Bhuvan Urgaonkar, George Kesidis, and Chita Das. Spock: Exploiting Serverless Functions for SLO and Cost Aware Resource Procurement in Public Cloud. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 199–208, 2019. https://doi.org/10.1109/CLOUD.2019.00043.

[50] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On Calibration of Modern Neural Networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[51] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025.

[52] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, page 243–254. IEEE Press, 2016. https://doi.org/10.1109/ISCA.2016.30.

[53] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7436–7456, 2022. https://doi.org/10.1109/TPAMI.2021.3117837.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, 2015. URL: https://arxiv.org/abs/1512.03385, arXiv:1512.03385.

[55] Zecheng He, Tianwei Zhang, and Ruby B. Lee. Attacking and Protecting Data Privacy in Edge–Cloud Collaborative Inference Systems. *IEEE Internet of Things Journal*, 8(12):9706–9716, 2021. https://doi.org/10.1109/JIOT.2020.3022358.

[56] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

[57] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[58] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, 2021. URL: https://arxiv.org/abs/2106.09685, arXiv:2106.09685.

[59] Haiyang Huang, Newsha Ardalani, Anna Sun, Liu Ke, Shruti Bhosale, Hsien-Hsin S. Lee, Carole-Jean Wu, and Benjamin Lee. Toward Efficient Inference for Mixture of Experts. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL: https://openreview.net/forum?id=stXtBqyTWX.

[60] Jim Huang and Philipp Landgraf. Remote Rendering for Real-time AR Applications at AWS Edge, 2024. URL: https://aws.amazon.com/blogs/industries/remote-rendering-for-real-time-ar-applications-at-aws-edge/.

[61] Jin Huang, Colin Samplawski, Deepak Ganesan, Benjamin Marlin, and Heesung Kwon. CLIO: Enabling Automatic Compilation of Deep Learning Pipelines across IoT and Cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, New York, NY, USA, 2020. Association for Computing Machinery. https://doi.org/10.1145/3372224.3419215.

[62] Kai Huang and Wei Gao. Real-Time Neural Network Inference on Extremely Weak Devices: Agile Offloading with Explainable AI. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, MobiCom '22, page 200–213, New York, NY, USA, 2022. Association for Computing Machinery. https://doi.org/10.1145/3495243.3560551.

[63] Yutao Huang, Yifei Zhu, Xiaoyi Fan, Xiaoqiang Ma, Fangxin Wang, Jiangchuan Liu, Ziyi Wang, and Yong Cui. Task Scheduling with Optimized Transmission Time in Collaborative Cloud-Edge Learning. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, 2018. https://doi.org/10.1109/ICCCN.2018.8487352.

[64] Zhenhua Huang, Shunzhi Yang, MengChu Zhou, Zheng Gong, Abdullah Abusorrah, Chen Lin, and Zheng Huang. Making Accurate Object Detection at the Edge: Review and New Approach. *Artificial Intelligence Review*, 55(3):2245–2274, 2022.

[65] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Serving Deep Learning Models in a Serverless Platform . In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 257–262, Los Alamitos, CA, USA, April 2018. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/IC2E.2018.00052, https://doi.org/10.1109/IC2E.2018.00052.

[66] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[67] Arthur S. Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A. Ferreira, Arpit Gupta, and Lisandro Z. Granville. AI/ML for Network Security: The Emperor Has No Clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1537–1551, New York, NY, USA, 2022. Association for Computing Machinery. https://doi.org/10.1145/3548606.3560609.

[68] Matthijs Jansen, Auday Al-Dulaimy, Alessandro V Papadopoulos, Animesh Trivedi, and Alexandru Iosup. The SPEC-RG Reference Architecture for the Compute Continuum. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 469–484. IEEE, 2023.

[69] Jananie Jarachanthan, Li Chen, Fei Xu, and Bo Li. AMPS-Inf: Automatic Model Partitioning for Serverless Inference with Cost Efficiency. In *50th International Conference on Parallel Processing*, ICPP 2021, New York, NY, USA, 2021. Association for Computing Machinery. https://doi.org/10.1145/3472456.3472501.

[70] Joohyung Jeon and Joongheon Kim. Privacy-Sensitive Parallel Split Learning. In *2020 International Conference on Information Networking (ICOIN)*, pages 7–9, 2020. https://doi.org/10.1109/ICOIN48656.2020.9016486.

[71] Congfeng Jiang, Tiantian Fan, Honghao Gao, Weisong Shi, Liangkai Liu, Christophe Cérin, and Jian Wan. Energy Aware Edge Computing: A Survey. *Computer Communications*, 151:556–580, 2020. URL: https://www.sciencedirect.com/science/article/pii/S014036641930831X, https://doi.org/https://doi.org/10.1016/j.comcom.2020.01.004.

[72] Jiawei Jiang, Shaoduo Gan, Bo Du, Gustavo Alonso, Ana Klimovic, Ankit Singla, Wentao Wu, Sheng Wang, and Ce Zhang. A systematic evaluation of machine learning on serverless infrastructure. *The VLDB Journal*, 33(2):425–449, September 2023. https://doi.org/10.1007/s00778-023-00813-0.

[73] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs, 2024. arXiv:2402.15627.

[74] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, Baltimore, MD, August 2018. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar.

[75] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, page 615–629, New York, NY, USA, 2017. Association for Computing Machinery. https://doi.org/10.1145/3037697.3037698.

[76] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. In *ICML*, 2019.

[77] Juyong Kim, Yookoon Park, Gunhee Kim, and Sung Ju Hwang. SplitNet: Learning to Semantically Split Deep Networks for Parameter Reduction and Model Parallelization. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1866–1874. PMLR, 06–11 Aug 2017. URL: https://proceedings.mlr.press/v70/kim17b.html.

[78] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. Technical Report 0, University of Toronto, 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[79] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in neural information processing systems*, 25, 2012.

[80] Karthik Kumar and Yung-Hsiang Lu. Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? *Computer*, 43(4):51–56, 2010. https://doi.org/10.1109/MC.2010.98.

[81] Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, New York, NY, USA, 2020. Association for Computing Machinery. https://doi.org/10.1145/3372224.3419194.

[82] Ya Le and Xuan Yang. Tiny ImageNet Visual Recognition Challenge. *CS 231N*, 7(7):3, 2015.

[83] Joo Chan Lee, Yongwoo Kim, SungTae Moon, and Jong Hwan Ko. A Splittable DNN-Based Object Detector for Edge-Cloud Collaborative Real-Time Video Inference. In *2021 17th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–8, 2021. https://doi.org/10.1109/AVSS52988.2021.9663806.

[84] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In Andreas Krause, Emma Brunskill,

Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 23–29 Jul 2023. URL: https://proceedings.mlr.press/v202/leviathan23a.html.

[85] Chao Li, Hongli Xu, Yang Xu, Zhiyuan Wang, and Liusheng Huang. DNN Inference Acceleration with Partitioning and Early Exiting in Edge Computing. In *Wireless Algorithms, Systems, and Applications: 16th International Conference, WASA 2021, Nanjing, China, June 25–27, 2021, Proceedings, Part I*, page 465–478, Berlin, Heidelberg, 2021. Springer-Verlag. https://doi.org/10.1007/978-3-030-85928-2_37.

[86] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457, 2019.

[87] En Li, Zhi Zhou, and Xu Chen. Edge Intelligence: On-Demand Deep Learning Model Co-Inference with Device-Edge Synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*, MECOMM'18, page 31–36, New York, NY, USA, 2018. Association for Computing Machinery. https://doi.org/10.1145/3229556.3229562.

[88] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Xiaobing Feng. Auto-tuning Neural Network Quantization Framework for Collaborative Inference Between the Cloud and Edge. In Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 402–411, Cham, 2018. Springer International Publishing.

[89] Jingtao Li, Adnan Siraj Rakin, Xing Chen, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. ResSFL: A Resistance Transfer Framework for Defending Model Inversion Attack in Split Federated Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10194–10202, June 2022.

[90] Shan Li, Weihong Deng, and JunPing Du. Reliable Crowdsourcing and Deep Locality-Preserving Learning for Expression Recognition in the Wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2852–2861, 2017.

[91] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and Quantization for Deep Neural Network Acceleration: A Survey. *Neurocomputing*, 461:370–403, 2021. URL: https://www.sciencedirect.com/science/article/pii/S0925231221010894, https://doi.org/https://doi.org/10.1016/j.neucom.2021.07.045.

[92] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration, 2024.

[93] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. Computation Offloading Toward Edge Computing. *Proceedings of the IEEE*, 107(8):1584–1607, 2019. https://doi.org/10.1109/JPROC.2019.2922285.

[94] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context, 2015. arXiv:1405.0312.

[95] Fang Liu, Guoming Tang, Youhuizi Li, Zhiping Cai, Xingzhou Zhang, and Tongqing Zhou. A Survey on Edge Computing Systems and Tools. *Proceedings of the IEEE*, 107(8):1537–1562, 2019. https://doi.org/10.1109/JPROC.2019.2920341.

[96] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*, 2019. URL: https://openreview.net/forum?id=S1eYHoC5FX.

[97] Haolin Liu, Sirui Liu, Saiqin Long, Qingyong Deng, and Zhetao Li. Joint Optimization of Model Deployment for Freshness-Sensitive Task Assignment in Edge Intelligence. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 1751–1760, 2024. https://doi.org/10.1109/INFOCOM52122.2024.10621314.

[98] Jianchun Liu, Hongli Xu, Yang Xu, Zhenguo Ma, Zhiyuan Wang, Chen Qian, and He Huang. Communication-Efficient Asynchronous Federated Learning in Resource-Constrained Edge Computing. *Comput. Netw.*, 199(C), apr 2022. https://doi.org/10.1016/j.comnet.2021.108429.

[99] Juncai Liu, Jessie Hui Wang, Chenghao Rong, Yuedong Xu, Tao Yu, and Jilong Wang. FedPA: An Adaptively Partial Model Aggregation Strategy in Federated Learning. *Comput. Netw.*, 199(C), apr 2022. https://doi.org/10.1016/j.comnet.2021.108468.

[100] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery. https://doi.org/10.1145/3300061.3300116.

[101] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-Real Net: Enhancing the Performance of 1-bit CNNs With Improved Representational Capability and Advanced Training Algorithm. In *Proceedings of the European conference on computer vision (ECCV)*, pages 722–737, 2018.

[102] Shuming Ma, Hongyu Wang, Shaohan Huang, Xingxing Zhang, Ying Hu, Ting Song, Yan Xia, and Furu Wei. BitNet b1.58 2B4T Technical Report. Technical Report arXiv:2504.12285, arXiv, Apr 2025. URL: https://arxiv.org/abs/2504.12285, https://doi.org/10.48550/arXiv.2504.12285.

[103] Yunlong Mao, Shanhe Yi, Qun Li, Jinghao Feng, Fengyuan Xu, and Sheng Zhong. Learning from Differentially Private Neural Activations with Edge Computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 90–102, 2018. https://doi.org/10.1109/SEC.2018.00014.

[104] Yoshitomo Matsubara and Marco Levorato. Neural Compression and Filtering for Edge-assisted Real-time Object Detection in Challenged Networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 2272–2279. IEEE, 2021.

[105] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges. *ACM Comput. Surv.*, 55(5), dec 2022. https://doi.org/10.1145/3527155.

[106] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. *arXiv preprint arXiv:1602.05629*, 2023. arXiv:1602.05629.

[107] Meta. MODEL_CARD, 2024. URL: https://github.com/meta-llama/llama-models/blob/main/models/llama3_1/MODEL_CARD.md.

[108] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 932–949, New York, NY, USA, 2024. Association for Computing Machinery. URL: https://doi-org.ezproxy.bu.edu/10.1145/3620666.3651335, https://doi.org/10.1145/3620666.3651335.

[109] Microsoft Learn. Azure Functions Overview: Hosting Options. https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview#hosting-options, 2025. Accessed: 2025-05-01.

[110] Robert B. Miller. Response Time in Man-Computer Conversational Transactions. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), page 267–277, New York, NY, USA, 1968. Association for Computing Machinery. https://doi.org/10.1145/1476589.1476628.

[111] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Ali Jalali, Ahmed Taha Taha Elthakeb, Dean Tullsen, and Hadi Esmaeilzadeh. Not all features are equal: Discovering essential features for preserving prediction privacy. In *Proceedings of the Web Conference 2021*, WWW '21, page 669–680, New York, NY, USA, 2021. Association for Computing Machinery. https://doi.org/10.1145/3442381.3449965.

[112] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Prakash Ramrakhyani, Ali Jalali, Dean Tullsen, and Hadi Esmaeilzadeh. Shredder: Learning Noise Distributions to Protect Inference Privacy. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 3–18, New York, NY, USA, 2020. Association for Computing Machinery. https://doi.org/10.1145/3373376.3378522.

[113] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522. USENIX Association, August 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/mishra.

[114] James O' Neill. An Overview of Neural Network Compression. *arXiv preprint arXiv:2006.03669*, 2020. URL: https://arxiv.org/abs/2006.

03669, https://doi.org/10.48550/ARXIV.2006.03669.

[115] Netflix. Netflix Empowers Remote Artistry with Low-Latency Workstations Using AWS Local Zones, 2024. URL: https://aws.amazon.com/solutions/case-studies/netflix-aws-local-zones-case-study/.

[116] Lucien K. L. Ng and Sherman S. M. Chow. Sok: Cryptographic neural-network computation. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 497–514, 2023. https://doi.org/10.1109/SP46215.2023.10179483.

[117] Samuel S. Ogden, Xiangnan Kong, and Tian Guo. PieSlicer: Dynamically Improving Response Time for Cloud-Based CNN Inference. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '21, page 249–256, New York, NY, USA, 2021. Association for Computing Machinery. https://doi.org/10.1145/3427921.3450256.

[118] OpenAI OpCo, LLC. Privacy policy. https://openai.com/policies/row-privacy-policy/, 2024. Published November 2024; Accessed: 2025-04-20.

[119] Kai Peng, Jiangtian Nie, Neeraj Kumar, Chao Cai, Jiawen Kang, Zehui Xiong, and Yang Zhang. Joint Optimization of Service Chain Caching and Task Offloading in Mobile Edge Computing. *Appl. Soft Comput.*, 103(C), May 2021. https://doi.org/10.1016/j.asoc.2021.107142.

[120] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently Scaling Transformer Inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

[121] Qwen. Speed Benchmark, 2024. URL: https://qwen.readthedocs.io/en/latest/benchmark/speed_benchmark.html.

[122] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020.

[123] Ali Raza, Abraham Matta, Nabeel Akhtar, Vasiliki Kalavri, and Vatche Isahagian. SoK: Function-as-a-Service: From An Application Developer's Perspective. In *Journal of Systems Research - Mar 2021*, 2021. URL: https://openreview.net/forum?id=VdWaMgaTKtX.

[124] Ali Raza, Zongshun Zhang, Nabeel Akhtar, Vatche Isahagian, and Ibrahim Matta. LIBRA: An Economical Hybrid Approach for Cloud Applications with Strict SLAs. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 136–146, 2021. https://doi.org/10.1109/IC2E52221.2021.00028.

[125] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger, 2016. arXiv:1612.08242.

[126] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement, 2018. arXiv:1804.02767.

[127] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016. arXiv:1506.01497.

[128] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. AI Accelerator Survey and Trends. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, 2021. https://doi.org/10.1109/HPEC49654.2021.9622867.

[129] Michael D Richard and Richard P Lippmann. Neural Network Classifiers Estimate Bayesian a posteriori Probabilities. *Neural computation*, 3(4):461–483, 1991.

[130] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. Llama: A Heterogeneous Serverless Framework for Auto-Tuning Video Analytics Pipelines. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, page 1–17, New York, NY, USA, 2021. Association for Computing Machinery. https://doi.org/10.1145/3472883.3486972.

[131] David E. Rumelhart and James L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pages 318–362. 1987.

[132] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. https://doi.org/10.1007/s11263-015-0816-y.

[133] Samsung. Samsung To Unveil New Vacuum Lineup That Redefines Home Cleaning With Enhanced AI at CES 2024, 2024. URL: https://news.samsung.com/us/samsung-

unveil-new-vacuum-lineup-redefines-home-cleaning-with-enhanced-ai-ces-2024/.

[134] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks, 2019. arXiv:1801.04381.

[135] Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. Can the Network Be the AI Accelerator? In *Proceedings of the 2018 Morning Workshop on In-Network Computing*, NetCompute '18, page 20–25, New York, NY, USA, 2018. Association for Computing Machinery. https://doi.org/10.1145/3229591.3229594.

[136] Kathakoli Sengupta, Zhongkai Shagguan, Sandesh Bharadwaj, Sanjay Arora, Eshed Ohn-Bar, and Renato Mancuso. UniLCD: Unified Local-Cloud Decision-Making via Reinforcement Learning, 2024. URL: https://arxiv.org/abs/2409.11403, arXiv:2409.11403.

[137] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 205–218. USENIX Association, July 2020. URL: https://www.usenix.org/conference/atc20/presentation/shahrad.

[138] Jiawei Shao, Yuyi Mao, and Jun Zhang. Learning Task-Oriented Communication for Edge Inference: An Information Bottleneck Approach. *IEEE Journal on Selected Areas in Communications*, 40(1):197–211, 2021.

[139] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*, 2017. URL: https://openreview.net/forum?id=B1ckMDqlg.

[140] Siemens. From City Theory to Smart Tech Reality, 2024. URL: https://www.siemens-advanta.com/whitepapers/smart-tech-reality.

[141] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2015. arXiv:1409.1556.

[142] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. A Survey on Mobile Augmented Reality With 5G Mobile Edge Computing: Architectures, Applications, and Technical Aspects. *IEEE Communications Surveys & Tutorials*, 23(2):1160–1192, 2021. https://doi.org/10.1109/COMST.2021.3061981.

[143] SKT. SKT and AWS Launch the First 5G Edge Cloud Service in Korea, 2024. URL: https://www.sktelecom.com/en/press/press_detail.do?page.page=1&idx=1494.

[144] Vladislav Sovrasov. ptflops: a FLOPs Counting Tool for Neural Networks in PyTorch Framework. https://github.com/sovrasov/flops-counter.pytorch, 2023. Access Date: 2023-12-16.

[145] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[146] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, 2016. https://doi.org/10.1109/ICPR.2016.7900006.

[147] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 328–339, 2017. https://doi.org/10.1109/ICDCS.2017.226.

[148] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. SplitFed: When Federated Learning Meets Split Learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 8485–8493, 2022.

[149] Joana Tirana, Spyros Lalis, and Dimitris Chatzopoulos. Estimating the Training Time in Single- and Multi-Hop Split Federated Learning. In *Proceedings of the 8th International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '25, page 37–42, New York, NY, USA, 2025. Association for Computing Machinery. https://doi.org/10.1145/3721888.3722096.

[150] Praneeth Vepakomma, Abhishek Singh, Otkrist Gupta, and Ramesh Raskar. NoPeek: Information leakage reduction to share activations in distributed deep learning. *arXiv preprint arXiv:2008.09161*, 2020. URL: https://arxiv.org/abs/2008.09161, https://doi.org/10.48550/ARXIV.2008.09161.

[151] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. 2021.

[152] G.K. Wallace. The JPEG Still Picture Compression Standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992. https://doi.org/10.1109/30.125072.

[153] Bo Wang, Changhai Wang, Wanwei Huang, Ying Song, and Xiaoyun Qin. A Survey and Taxonomy on Task Offloading for Edge-Cloud Computing. *IEEE Access*, 8:186080–186101, 2020.

[154] Bo Wang, Changhai Wang, Wanwei Huang, Ying Song, and Xiaoyun Qin. A Survey and Taxonomy on Task Offloading for Edge-Cloud Computing. *IEEE Access*, 8:186080–186101, 2020. https://doi.org/10.1109/ACCESS.2020.3029649.

[155] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing Federated Learning on Non-IID Data with Reinforcement Learning. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 1698–1707, 2020. https://doi.org/10.1109/INFOCOM41043.2020.9155494.

[156] Kuan-Chieh Wang, YAN FU, Ke Li, Ashish Khisti, Richard Zemel, and Alireza Makhzani. Variational Model Inversion Attacks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9706–9719. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper/2021/file/50a074e6a8da4662ae0a29edde722179-Paper.pdf.

[157] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. SkipNet: Learning Dynamic Routing in Convolutional Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[158] Yingchao Wang, Chen Yang, Shulin Lan, Liehuang Zhu, and Yan Zhang. End-Edge-Cloud Collaborative Computing for Deep Learning: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*, pages 1–1, 2024. https://doi.org/10.1109/COMST.2024.3393230.

[159] Yue Wang, Jianghao Shen, Ting-Kuei Hu, Pengfei Xu, Tan Nguyen, Richard G. Baraniuk, Zhangyang Wang, and Yingyan Lin. Dual Dynamic Inference: Enabling More Efficient, Adaptive and Controllable Deep Inference. *IEEE Journal of Selected Topics in Signal Processing*, 2020. URL: https://par.nsf.gov/biblio/10159763, https://doi.org/10.1109/JSTSP.2020.2979669.

[160] Zhiyuan Wang, Hongli Xu, Yang Xu, Zhida Jiang, and Jianchun Liu. CoopFL: Accelerating Federated Learning with DNN Partitioning and Offloading in Heterogeneous Edge Computing. *Computer Networks*, 220:109490, 2023.

[161] Campbell Webb. Unleashing the Power of Innovation with Public Cloud, 2024. URL: https://blog.workday.com/en-us/unleashing-the-power-innovation-with-public-cloud.html.

[162] Wenqi Wei and Ling Liu. Trustworthy Distributed AI Systems: Robustness, Privacy, and Governance. *ACM Comput. Surv.*, February 2024. Just Accepted. https://doi.org/10.1145/3645102.

[163] Maciej Woł czyk, Bartosz Wójcik, Klaudia Bał azy, Igor T Podolak, Jacek Tabor, Marek Śmieja, and Tomasz Trzcinski. Zero Time Waste: Recycling Predictions in Early Exit Neural Networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 2516–2528. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper/2021/file/149ef6419512be56a93169cd5e6fa8fd-Paper.pdf.

[164] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[165] Shuzhao Xie, Yuan Xue, Yifei Zhu, and Zhi Wang. Cost Effective MLaaS Federation: A Combinatorial Reinforcement Learning Approach. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, page 2078–2087. IEEE Press, 2022. https://doi.org/10.1109/INFOCOM48880.2022.9796701.

[166] Xiong Xiong, Kan Zheng, Lei Lei, and Lu Hou. Resource Allocation Based on Deep Reinforcement Learning in IoT Edge Computing. *IEEE Journal on Selected Areas in Communications*, 38(6):1133–1146, 2020. https://doi.org/10.1109/JSAC.2020.2986615.

[167] Yuanjia Xu, Heng Wu, Wenbo Zhang, and Yi Hu. EOP: Efficient Operator Partition for Deep Learning Inference over Edge Servers. In *Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE 2022,

page 45–57, New York, NY, USA, 2022. Association for Computing Machinery. https://doi.org/10.1145/3516807.3516820.

[168] Zhao Yang, Shengbing Zhang, Ruxu Li, Chuxi Li, Miao Wang, Danghui Wang, and Meng Zhang. Efficient Resource-Aware Convolutional Neural Architecture Search for Edge Computing with Pareto-Bayesian Optimization. *Sensors*, 21(2), 2021. URL: https://www.mdpi.com/1424-8220/21/2/444, https://doi.org/10.3390/s21020444.

[169] Ziqi Yang, Jiyi Zhang, Ee-Chien Chang, and Zhenkai Liang. Neural Network Inversion in Adversarial Setting via Background Knowledge Alignment. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 225–240, New York, NY, USA, 2019. Association for Computing Machinery. https://doi.org/10.1145/3319535.3354261.

[170] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. Deep Compressive Offloading: Speeding up Neural Network Inference by Trading Edge Computation for Network Latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, page 476–488, New York, NY, USA, 2020. Association for Computing Machinery. https://doi.org/10.1145/3384419.3430898.

[171] Yuanshun Yao, Zhujun Xiao, Bolun Wang, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Complexity vs. Performance: Empirical Analysis of Machine Learning as a Service. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 384–397, New York, NY, USA, 2017. Association for Computing Machinery. https://doi.org/10.1145/3131365.3131372.

[172] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving. In *Proceedings of the 8th Conference on Machine Learning and Systems (MLSys)*, Santa Clara, CA, USA, 2025. To appear. URL: https://arxiv.org/abs/2501.01005, https://doi.org/10.48550/arXiv.2501.01005.

[173] Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-Friendly Differential Privacy Library in PyTorch, 2022. arXiv:2109.12298.

[174] Li Yuan, Francis EH Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting Knowledge Distillation via Label Smoothing Regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[175] Jiale Zhang, Yanchao Zhao, Junyu Wang, and Bing Chen. FedMEC: Improving Efficiency of Differentially Private Federated Learning via Mobile Edge Computing. *Mobile Networks and Applications*, 25(6):2421–2433, Dec 2020. https://doi.org/10.1007/s11036-020-01586-4.

[176] Linfeng Zhang, Chenglong Bao, and Kaisheng Ma. Self-Distillation: Towards Efficient and Compact Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4388–4403, 2022. https://doi.org/10.1109/TPAMI.2021.3067100.

[177] Zixin Zhang, Fan Qi, and Changsheng Xu. Enhancing Storage and Computational Efficiency in Federated Multimodal Learning for Large-Scale Models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 59685–59699. PMLR, 21–27 Jul 2024. URL: https://proceedings.mlr.press/v235/zhang24az.html.

[178] Zongshun Zhang, Rohan Kumar, Jason Li, Lisa Korver, Anthony Byrne, Gianluca Stringhini, Ibrahim Matta, and Ayse Coskun. PraxiPaaS: A Decomposable Machine Learning System for Efficient Container Package Discovery. In *12th IEEE International Conference on Cloud Engineering*, 2024.

[179] Zongshun Zhang, Andrea Pinto, Valeria Turina, Flavio Esposito, and Ibrahim Matta. Privacy and Efficiency of Communications in Federated Split Learning. *IEEE Transactions on Big Data*, 9(5):1380–1391, 2023. https://doi.org/10.1109/TBDATA.2023.3280405.

[180] Helong Zhou, Liangchen Song, Jiajie Chen, Ye Zhou, Guoli Wang, Junsong Yuan, and Qian Zhang. Rethinking Soft Labels for Knowledge Distillation: A Bias-Variance Tradeoff Perspective. *arXiv preprint arXiv:2102.00650*, 2021.

[181] Hongbo Zhou, Weiwei Zhang, Chengwei Wang, Xin Ma, and Haoran Yu. BBNet: A Novel Convolutional Neural Network Structure in Edge-Cloud Collaborative Inference. *Sensors*, 2021.

[182] Huan Zhou, Zhenning Wang, Hantong Zheng, Shibo He, and Mianxiong Dong. Cost Minimization-Oriented Computation Offloading and Service Caching in Mobile Cloud-Edge Computing: An A3C-Based Approach. *IEEE Transactions on Network Science and Engineering*, 10(3):1326–1338, 2023. https://doi.org/10.1109/TNSE.2023.3255544.

[183] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. BERT Loses Patience: Fast and Robust Inference with Early Exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020.