Systematizing Inference Placement for Deep Learning across Edge and Cloud Platforms: A Multi-Objective Optimization Perspective

Zongshun Zhang
Department of Computer Science
Boston University

Ibrahim Matta
Department of Computer Science
Boston University

Abstract

Edge intelligent applications like VR/AR and language model based chatbots have become widespread with the rapid expansion of IoT and mobile devices. However, constrained edge devices often cannot serve the increasingly large and complex deep learning (DL) models. To mitigate these 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 contextualizes the state-of-the-art model offloading methods and model adaptation techniques by studying their implication to a multi-objective optimization comprising inference latency, data privacy, and resource monetary cost.

1 Introduction

In recent decades, large volumes of data have been generated on mobile and Internet-of-Things (IoT) devices. While cloud computing resources remain more flexible in scaling and management than edge resources, edge computing can mitigate transmission bottlenecks over the wide-area network connecting users to the cloud [144]. As a result, recent intelligent systems increasingly offload user applications across a continuum spanning personal devices, edge servers, and cloud infrastructures [70, 102, 105]. These systems leverage resource orchestration services at multiple levels of the stack, optimizing for latency, privacy, and monetary cost. As exemplified in Fig. 1, a Machine Learning-as-a-Service (MLaaS) system can provision resources across this continuum using container, Virtual Machine (VM), or Serverless function to serve a Machine Learning (ML) request x^1.

ML applications have diverse performance requirements and face varying resource constraints. For instance, mobile and IoT applications, including object recognition in housekeeping AIoT devices [154] and localization in autonomous cars [5], are limited by energy consumption [73] or rely on emerging infrastructures like 5G/6G base stations and smart city networks [39, 161, 163].

At the same time, the high monetary cost of computational resources for AI/ML training and inference creates a barrier, especially for research institutions and smaller companies. In contrast, large firms can afford to build extensive Deep Learning (DL) clusters. For example, pre-training LLaMA-3.1-8B (LLaMA-3.1-405B) requires 1.46 million (30.84 million) in H100-80GB GPU hours [120], and ByteDance operates a cluster of over 10,000 NVIDIA Ampere GPUs for its Large Language Model (LLM) workloads [78].

Building an infrastructure that is monetary cost-efficient, latency-optimized, and privacy-aware for architecture-optimized models has become a critical area of research [47, 54, 183]. For an inference task, given a Neural Network (NN) model and a data source, we can split the model and provision its parts on cloud services, edge servers, or client devices using resource orchestration services (e.g., Virtual Machines, Containers, etc.) As shown in Fig. 1, model decomposition creates either an ensemble of independent submodels, or a tandem of dependent submodels (partitions of layers, given a multi-layered ML model). We consider an MLaaS system that places these submodels across the continuum of user devices, edge, and cloud using orchestration methods such as containers, VMs, and serverless platforms.

In this survey, we systematize model-decomposition research from an optimization perspective. The common formulation established in Section 3 helps contextualize the breadth of related work. This Systematization of Knowledge takes a holistic view of multiple objectives, including inference latency, source data privacy, and monetary cost, whose integration has not been comprehensively addressed in related surveys on quantization [129], weight pruning [100], distillation [50], privacy preserving distributed learning [186], and monetary cost-based resource provisioning using edge and cloud [28]. By exploring interactions among these optimization objectives, we highlight the trade-offs inherent in existing ML model offloading techniques.

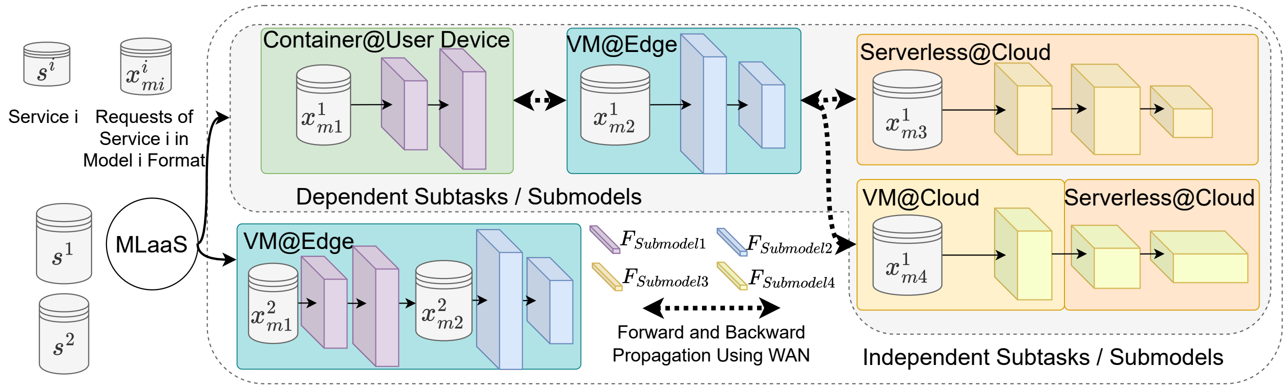

Figure 1: A MLaaS System Offloads Service/Applications s^i across the Continuum of Device, Edge and Cloud Resources Using Different Resource Orchestration Platforms.

1.1 The Cost Model of Machine Learning-as-a-Service

Machine Learning-as-a-Service (MLaaS) is a resource orchestration model that provides compute resources for ML training and inference on a pay-per-use basis. As exemplified in Fig. 1, an MLaaS system can provision resources across the continuum of user devices, edge, and cloud resources. For example, for an ML service s^1 which prioritizes resource cost while maintaining low latency, an MLaaS framework can deploy the ML model in the cloud—leveraging the high parallelism of cloud resources—and offload the lightweight, heavily used portions to edge or user devices using containers or VMs. On the other hand, for an ML service s^2 that prioritizes the privacy of source data, an MLaaS broker could provision the model using particular secure computation resources like client devices or VMs.

The MLaaS system abstracts away infrastructure management, so users only pay for what they consume. Existing MLaaS systems provide managed services in the cloud and at the edge. Depending on how much configuration they allow, MLaaS systems expose resources at various levels of abstraction [198]. For example, AWS provides a set of *AI services*, including Amazon Rekognition [11], which hide lower-level details like the ML models and computation resources from users. In contrast, their *ML services*, such as Amazon Sagemaker [12], allow users to define models, data sources, and resource orchestration across VM instances, serverless instances, S3 object storage, etc. Sagemaker Edge [13] further extends this by deploying NN models on user-owned IoT devices and gathering data for inference and retraining.

MLaaS is vital for many ML workloads. Although modern Large Language Model (LLM) training and inference demand enormous compute power and high inter-node bandwidth, typically available only in private GPU clusters [78], small teams can leverage parameter-efficient fine-tuning and inference on smaller models within public cloud or edge-based MLaaS environments. For example, fine-tuning a $65B$ parameter model with a Low-Rank Adapter (LoRA) requires only a single $48GB$ GPU for less than 24 hours [29]. Likewise, running inference on a $Qwen2-7B-Instruct$ model using one $A100-80GB$ GPU achieves 41.20 tokens per second [140]. Therefore, instead of building and maintaining their own compute clusters, smaller companies can act as MLaaS brokers, hosting decomposed model components – e.g., LoRA adapters for LLMs or traditional Deep Neural Networks (DNNs) – in virtualized cloud or edge environments to serve customers and support internal R&D.

Recently, organizations such as Adobe [2, 6] and Workday [185], have launched hybrid-cloud MLaaS platforms. They combine public-cloud resources (VMs or containers on AWS, Azure, or GCP) with private clusters (at the edge close to the company users). This approach involves trade-offs and opportunities around service latency guarantee, user data privacy preservation, and savings in monetary cost of computation resources.

For example, provisioning $H100$ GPUs in the cloud offers fast processing, but incurs a high monetary resource cost and may increase data transmission time compared to private clusters or other edge resources located close to end users. Conversely, allocating computation resources in the cloud instead of processing data securely at the edge, comprising the 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 application-level objectives (accuracy, latency, privacy, and monetary cost) can be relaxed to improve other ones. For example, federated learning on medical data has low tolerance to data leakage. To meet this requirement, an MLaaS broker might run only a sensitive partition (e.g., shallow layers of a model) at the edge, so the raw data never leaves the device, while offloading the remaining partitions to the cloud. Due to the limited edge capacity and transmission between the edge and cloud, the training time can be longer and overall monetary cost could be higher. In contrast, clear data-sharing agreements can mitigate privacy concerns for less sensitive tasks and potentially allow the use of cheaper cloud resources [49, 134].

Investigating such tradeoffs in both traditional DNNs and LLMs yields valuable guidance for future MLaaS system design, helping smaller companies and institutions deploy large-scale ML systems with low latency, high privacy guarantee, and low monetary cost of resources. 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. This section highlights recent work and open challenges in these dimensions via resource provisioning and model decomposition, leading to the contributions of this survey.

What are the limitations of existing MLaaS systems for ML Inference? Existing MLaaS services do not yet incorporate much of the existing research on model decomposition and resource provisioning to improve latency [171], privacy [131], and service cost [30, 190]. Sagemaker Edge compiles NN models to utilize the client's hardware architecture and memory access patterns for optimal ML inference speed [14], which represents only a small slice of possible model adaptations. In contrast, integrating model decomposition, assigning each submodel to devices, edge, or cloud nodes, and provisioning compute, memory, and network resources for each, has not yet been adopted by mainstream MLaaS offerings.

Therefore, our research investigates model architecture optimization and resource provisioning strategies that could become part of future MLaaS services. In particular, we focus on model offloading in ML inference, including both independent and dependent model decomposition and resource orchestration with cloud and edge provisioning services, needed to meet strict latency, privacy, and monetary cost objectives.

1.2 Opportunities and Challenges of DL Task offloading during Inference

Why offloading Deep Learning (DL) tasks? ML applications that gather large volumes of data and use complex deep learning models often require low latency to meet quality-of-service (QoS) targets [56, 66, 109, 178]. Offloading selected layers to an edge or cloud tier can: (1) minimize inference latency by leveraging cloud compute capacity; (2) enhance source data privacy; and (3) reduce resource monetary cost via pay-per-use pricing models [158]. In this section, we examine one example: offloading user-interactive, latency-sensitive applications, such as Augmented Reality (AR) with partitions of a multi-layered ML model (dependent submodels), across edge and cloud environments to optimize inference latency and protect user data privacy.

For the threat model, we assume an **honest-but-curious** adversary who can:

- monitor activations x_{interm} in transit between the edge and cloud;
- train an offline reconstructor (auto-encoder NN) $\mathcal{R}_{\text{MIA}}$ on public data;
- output the reconstructed input data from observed intermediate data, i.e., $\hat{x}_{input} = \mathcal{R}_{\text{MIA}}(x_{interm})$, evaluated by mean-squared reconstruction error or misclassification rate on a target classifier.

This aligns with prior model inversion attack (MIA) and prompt inversion attack (PIA) formulations [33, 42, 113, 139].

Tables 1, 2, and 3 summarize representative edge, cloud, and hybrid edge-cloud results that motivate our offloading study. As shown in Table 1, recent lightweight DNN models used in AR applications on edge devices, such as Raspberry Pi or smartphones, often struggle to meet the 30 frames-per-second (FPS) video requirements or 100ms human-sensible end-to-end (frame refresh) latency target [24, 37, 112, 124, 174]. Edge platforms are constrained by limited compute power [40, 149], bandwidth [119], battery capacity [86], and memory [108], making it difficult to sustain high performance.

Cloud resources can provide the extra processing capacity needed for demanding ML workloads, as shown in Table 2. However, relying on the public cloud raises privacy concerns [210] and introduces transmission bottlenecks over the Internet [40, 212], which prohibit transmitting source data to the cloud for various ML tasks.

To combine the advantages of both edge and cloud resources, recent research has focused on partitioning ML tasks and provisioning resources across the cloud and the edge, with customer data residing on edge servers or user devices to minimize data leakage. In this approach, a portion of the ML task is performed on the client devices. Only essential hidden/latent variables required for high-accuracy inference 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. Furthermore, to improve data privacy during inference and defend against MIA attacks, previous work introduces privacy-preserving training steps, including adding privacy-aware loss terms and differential privacy approaches to gauge the hidden variables that adversaries can leverage.

Then, to compensate for the potential added latency, this paradigm often ensures that data transmission occurs only when necessary, for example, by using early exits [81] or dynamic region of interest encoding [110]. Specifically, a multi-layered/partitioned ML model can be augmented with "internal classifiers" that can produce early output (prediction) and may not need to process data through all layers/partitions. Similarly, the input data can be compressed/cropped based

Model	Device	End-to-End	Pred	Task	Ref
MNetV3 [60]	Rasp Pi $4B+$	$595ms$	79.23% accuracy	$48*48$-pixel RAF-DB [97]	[66]
MNetV2 [155]	Rasp Pi $4B+$	$3571ms$	81.16% accuracy	$48*48$-pixel RAF-DB [97]	[66]
MNetV2 +SSDLite [155]	Pixel 1	$162ms$	22.1% mAP	COCO [103]	[60]
MNetV3 +SSDLite [155]	Pixel 1	$137ms$	22.0% mAP	COCO [103]	[60]
YOLOv3 [147]	Pixel 2	$4500ms$	40% IOU	Imagenet Video [85]	[20]
Tiny-YOLO [146]	Pixel 2	$1200ms$	40% IOU	Imagenet Video [85]	[20]

Table 1: DNN performances at the edge in recent work

Model	Hardware	Processing	Pred	Task	Ref
YOLOv4-608 [147]	V100	$16.1ms$	43.5% COCOmAP	COCO [103]	[21]
YOLOv3-608 [147]	Titan X	$57.9ms$	33% COCOmAP	COCO [103]	[147]
YOLOv2-544 [146]	Titan X	NA	21.6% COCOmAP	COCO [103]	[147]

Table 2: DNN performances in the cloud in recent work

Model	Edge	Cloud	End-to-End	Pred	Bandwidth	Task	Ref
Faster R-CNN [148]	Jetson TX2	Titan XP	$34.56ms$	70% IoU	82.8Mbps	Xiph [192]	Baseline [110]
Faster R-CNN [148]	Jetson TX2	Titan XP	$22.96ms$	75.8% IoU	276Mbps	Xiph [192]	Baseline [110]
Faster R-CNN [148]	Jetson TX2	Titan XP	$17.23ms$	86.4% IoU	82.8Mbps	Xiph [192]	DRE +PSI +MvOT [110]
Faster R-CNN [148]	Jetson TX2	Titan XP	$15.52ms$	91.1% IoU	276Mbps	Xiph [192]	DRE +PSI +MvOT [110]

Table 3: End-to-end DNN performances combining edge and cloud resources

on the region of interest (RoI) to minimize data transmission. As shown in Table 3, some existing approaches based on this design achieve low latency and high model accuracy.

While the examples above focus on dependent submodels, there are also opportunities and challenges for independent submodel-based applications in achieving low latency, high privacy, and low monetary cost. For example, offloading different independent submodels from the ensemble model to multiple VMs or serverless function instances enables efficient incremental training and minimizes inference time via parallel execution. In this paper, we explore similar trade-offs among latency, privacy, and monetary cost by leveraging diverse model decomposition techniques and resource-orchestration services both at the edge and in the cloud.

Challenges and Explorations in ML task offloading. Finding an optimal offloading plan for ML applications is not trivial. A naïve offloading plan can result in long transmission and processing delays, privacy breaches (Model Inversion Attack), or resource under- or over-provisioning. To capture these trade-offs, we formulate an optimization problem that balances latency, privacy, and monetary cost based on 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 remains limited. Furthermore, while some existing surveys [118, 183] provide valuable insights, they often lack comprehensive discussions on source data privacy in distributed inference systems.

In this work, grounded in a multi-objective optimization framework, we categorize prior work and study multi-objective optimization accounting for interactions among latency, privacy, and monetary cost. To narrow the scope of our literature review, the survey emphasizes model-inversion attacks [42, 180, 196] and cloud pricing models [158] for deep-neural-network inference pipelines composed of multiple machine-learning models.

Our main contributions are:

- A Systematization of Knowledge for ML task offloading grounded in recent research, adopting a multi-objective optimization perspective and showing how specific related work (use cases) maps to the holistic formulation.
- A series of case studies analyzing interactions among optimization objectives (e.g., IaaS vs. FaaS latency–cost trade-offs, privacy–latency tensions, and strategies for jointly balancing latency, cost, and privacy), demonstrating how different ML model offloading tradeoffs manifest themselves.
- Identification of previously unstudied combinations of objectives under alternative threat models (e.g., prompt-inversion attacks in large language models), pricing models (e.g., serverless fine-grained billing vs. VM coarse-grained billing), and application domains.

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. 2.1.3, and Monetary Cost ($) in Sec. 2.1.2. Then, in Sec. 3, we formulate the optimization problem for Latency (Sec. 3.1), Privacy (Sec. 3.3) and Monetary Cost (Sec. 3.2). In Sec. 4, we detail adaptive learning methods to deploy a DNN model across the spectrum of cloud, edge, and client resources by optimizing Latency (Sec. 4.1), Privacy (Sec. 4.3), and Monetary Cost (Sec. 4.2). Next, Sec. 5 discusses the interactions between optimization objectives in the formulation given certain use cases introduced by related works. Sec. 6 discuss the open issues. And Sec. 7 concludes the paper.

2 Problem Definition

In this section, we narrow down the specific challenges in the machine learning (ML) model offloading problem that this paper addresses by examining the deep neural network (DNN) offloading scenario (dependent submodels). Recent studies have explored offloading a Deep Neural Network (DNN) model across the core cloud, edge, and client devices to satisfy resource constraints and privacy guarantees. Given dynamic environments, each inference request can adaptively route through model partitions on the most capable resources to minimize latency and maintain privacy constraints in a cost-efficient manner.

However, partitioning the NN model, introduces new challenges. Between model partitions, hidden variables and gradients transmitted during forward and backward propagation add additional transmission overhead [40, 197, 212] and cause client data leakage [175, 210]. Meanwhile, byproducts of running on the edge, for example, extra processing delays [63, 117] and energy consumption [81], should be minimized.

2.1 DNN Offloading Challenges

Existing MLaaS systems manage cloud resources [11] or user devices to run DL jobs [13]. Meanwhile, cloud-managed edge computing resources, including AWS Local Zones [16] and Wavelength [17], and edge ML model optimizers have become important building blocks for ML services used by companies such as Holo-Light [62], Netflix [130], and SKT [164]. With AWS Sagemaker Edge [13] and AWS Greengrass [15], 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 review the challenges of achieving high DNN performance when a DL model is partitioned between cloud, edge, and user devices.

2.1.1 Latency

In recent ML applications, including real-time object detection in AR/VR applications and LLM based ChatBots, latency has become an important concern considering the interactive manner of such applications, i.e, $\leq 100ms$ or $\geq 30fps$ [37, 112, 124, 174]. However, it is non-trivial to achieve low latency given the resource demand of the highly-parametrized models. In this section, we examine the challenges of minimizing ML inference latency.

The time spent in a distributed ML inference system can be decomposed into transmission and processing delays. When large volumes of data are sent between model partitions, transmission overhead can dominate inference latency [108, 128, 210]. Meanwhile, offloading too many 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] must be sent between partitions to complete forward propagation, often over bandwidth-constrained internet in IoT or mobile device settings. In traditional DNN models (e.g.,

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

Optimization Formulation	Monetary Cost ($)	Latency	Privacy	DL	Placement	Scope	Inference	Training	Reference
✓	✓	✓	✓	✓	✓	Edge Devices & Edge & Cloud	✓	○	(Our Work)
✗	✓	✓	✗	✓	✓	Graph in Mobile & Cloud	✓	✓	2020 [179]
✗	✗	✓	✓	✓	✓	AIoT & Edge & Cloud	✓	✓	2021 [22]
✗	✗	✓	○	✓	✓	Early Exit in Mobile & Cloud	✓	✓	2022 [118]
✗	✗	✓	✓	✓	✓	End Device & Edge & Cloud	✓	✓	2023 [34]
✗	✗	✓	○	✓	✓	End Device & Edge & Cloud	✓	✓	2024 [183]
✗	✓	✓	○	✓	✗	LLM Prompt Leakage	✓	✓	2024 [4]

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.

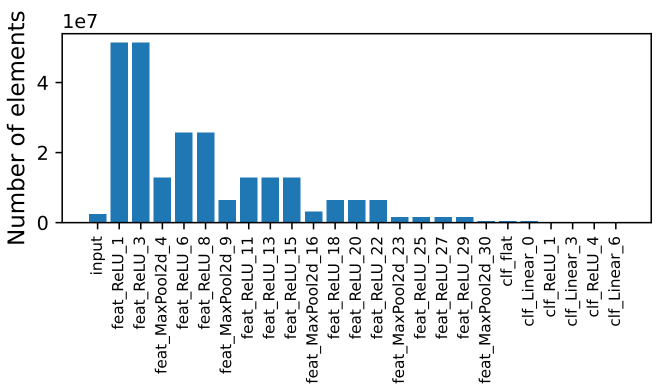

Figure 2: Hidden variable sizes of VGG16 with CiFAR-10 and batch size of 16.

convolutional layers, fully connected layers), partitioning is often by layers. Because intermediate representations have fixed shapes, transmission size can be estimated accordingly, enabling relatively simple offloading strategies. As shown in Fig. 2, we profiled hidden variable sizes for the VGG-16 model [162] on CiFAR-10 [84] with a batch size of 16. The x-axis denotes the layer where the model is split, with the "head" (input through the split layer) running on a client device and the "tail" running on an edge or cloud server. The y-axis reports the corresponding output size. Different splitting layers yield different transmission volumes. Therefore, an appropriate splitting point is essential for short delay [81].

For transformer-based models, auto-regressive token generation is stateful. When decoder blocks are offloaded, the KV cache of previously generated tokens must also be migrated, and its size grows with sequence length. This state migration complicates transmission estimation, since both the intermediate activations and model state must be transferred. Several recent works examine model splitting and offloading under state migration constraints [80, 128].

To reduce communication overhead, prior work has explored compressing both intermediate data and model components. Bottleneck networks (e.g., autoencoders) placed at the splitting point can selectively encode salient features for transmission [40, 64, 65, 117, 159, 197, 212]. Model-trimming approaches such as distillation [59] and quantization [63, 68, 111, 156] also reduce the size of hidden variables and gradients. Sparse or early-exit models activate only a subset of parameters during inference. For example, Internal Classifiers (ICs) can terminate forward propagation within a single partition when sufficient confidence is reached, eliminating the need to transmit intermediate activations [87, 171]. In such cases, downstream feature extraction and communication are bypassed entirely.

Processing. Another challenge for offloaded deep learning systems is the limited processing capacity of edge and client devices. As shown in Table 1, devices such as Raspberry Pi [66] and mobile phones [20, 60] often struggle to meet the latency or accuracy requirements demanded by machine learning applications.

To address these constraints, prior work has explored a variety of model adaptations, including quantization [100], pruning [57, 106, 184, 195], and knowledge distillation [59]. These techniques reduce model complexity, allowing inference tasks deployed on edge devices or cloud-assist pipelines to satisfy Quality-of-Service (QoS) objectives. Beyond architectural adaptations, other research leverages cloud to assist edge intelligence applications [30,87,114,172]. However, this approach introduces additional challenges related to privacy and communication overhead [64, 203].

2.1.2 Monetary Cost

As cloud infrastructure has evolved, providers have introduced services beyond VMs that can be more cost-efficient. For example, cloud offerings across providers exhibit individualized cost models [137] and varying performance characteristics [190]. These include Function-as-a-Service (FaaS) and Container-as-a-Service (CaaS) clusters in the cloud [8, 9], as well as AWS Lambda@Edge and Local Zones at the edge [10, 16], which provide fine-grained billing [158]. An MLaaS system should therefore adaptively configure both its runtime environment and model architecture to optimize cost and performance across the device–edge–cloud continuum.

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 [142, 143]. Related works have explored dynamically directing workload to a deep NN in the cloud and a shallow NN at the edge for cost savings [30]. Other works deploy NN partitions using Function-as-a-Service (FaaS) [71]. 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 NN adaptations can enable resource provisioning for individual NN layers, achieving cost-effective QoS tracking. By incorporating internal classifiers [82, 181] or neuron skipping methods [83], 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 [143]. Specifically, low-workload layers can be provisioned on demand with FaaS platforms [9, 10, 158] 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 [82, 125]. Consequently, these less frequently used deep layers are well-suited for FaaS.

2.1.3 Privacy

The 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 are not sent over the network. However, data breaches can occur as adversaries exploit the information in the intermediate data.

Recent works [116, 175, 203, 210] discuss the use of Auto-Encoders [152, 173] to reconstruct the source data from the intermediate data sent from the edge to the cloud during forward propagation for a deep neural network (DNN). Such vulnerability can be exploited by the model inversion attack (MIA) [42, 180, 196]. 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 to approximate matrix multiplication inversions.

As an example, for an ML application which infers the number in MNIST images of hand-written digits [89], when the DNN is partitioned and offloaded to different edge or cloud resources, the hidden variables transmitted over the wire can be captured by an honest but curious adversary, as a man-in-the-middle attack, leading to data leakage. To reveal the source data using the intermediate data (hidden variables), the adversary can train an Auto-Encoder model that can faithfully reconstruct a Kuzushiji-MNIST [26] dataset that has similar patterns in hand-writing and use the decoder NN to invert hidden variables to the source data. Recent work discovered a similar attack that reveals the user prompt to an LLM leveraging the hidden variables transmitted over the network, namely Prompt Inversion Attack (PIA) [113, 139], which further emphasizes the pervasive nature of such attacks in real-world applications. The formal definition of our attack model is presented in Sec. 3.3.1.

This attack is not merely theoretical but poses a practical threat in real-world settings for two key reasons. First, the attacker's decoder architecture can be flexible and does not need to precisely mirror the client-side model to be effective [96, 210]. Second, the proliferation of powerful open-source LLMs [54, 120] creates a significant vulnerability, particularly in collaborative edge-cloud inference systems [30]. In such a setup, an honest-but-curious intermediate node can leverage the publicly known weights of a base model. Since many deployed models are fine-tuned versions of these public foundation models, the adversary can exploit this shared architectural knowledge to train a highly effective reconstructor (e.g., an Auto-Encoder [96]) and recover sensitive source data from the intermediate activations.

Privacy-preserving methods for model and user data are critical for an MLaaS system. One approach involves encryption [79, 127], which can cause significant slowdowns [31, 113, 125, 139]. As shown in Table 5, without encryption, running VGG-16 on ImageNet takes 14.5 ms per inference on an NVIDIA Titan Xp GPU. However, when using FALCON homomorphic encryption on CPUs, the same task takes 12,960 ms [125].

The more popular privacy-preserving approach, however, mitigates MIA attacks by focusing on the training phase to build privacy-preserving models in the first place, rather than adapting the model during inference time. Such methods introduce a secondary loss function, e.g., distance correlation [72, 175, 210], to constrain the similarity between intermediate data and source data during model training. Similar

Model	Hardware	Processing	Task	Privacy	Ref
VGG-16 [176]	P100	$57ms$	Tiny ImageNet [88]	Plaintext	[176]
VGG-16 [176]	CPU	$1,300ms$	Tiny ImageNet [88]	Plaintext	[176]
VGG-16 [176]	CPU(LAN)	$40,000ms$	Tiny ImageNet [88]	SMPC(FALCON)	[176]
VGG-16 [176]	CPU(LAN)	$59,000ms$	Tiny ImageNet [88]	SMPC(FALCON)	[176]
VGG-16 [125]	Titan Xp	$14.5ms$	ImageNet [153]	Plaintext	[125]
VGG-16 [125]	CPU	$12,960ms$	ImageNet [153]	SMPC(FALCON)	[125]
VGG-16 [125]	Titan Xp	$14.5ms$	ImageNet [153]	Plaintext(Cloak)	[125]

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

works also incorporate an Auto-Encoder to model training, using reconstruction error as privacy metric [96]. Furthermore, previous works utilize DNN pruning with *masks* to remove mutual information between the source and intermediate data [31, 125].

Similarly, other privacy-preserving methods, instead of adjusting the loss function, apply perturbations to intermediate data during training [116, 203]. In these approaches, the intermediate activations retain minimal sensitive information, while the cloud-side neural network learns to extract the key features required for inference.

3 Problem Formulation

We identify three key challenges in Deep Neural Network offloading – covering both model decomposition and resource provisioning – across the device-edge-cloud continuum, each tied to a different performance objective:

- Latency: Minimizing end-to-end inference time requires partitioning the neural network such that local computation and network transmission are jointly balanced.
- Monetary Cost: For sparsely activated models, submodels can be mapped to the most cost-effective cloud services using fine-grained resource-provisioning strategies, while still meeting Service Level Objectives (SLOs).
- Privacy: Hidden variables transmitted over the network can be exploited by adversaries (e.g., via autoencoder–based inversion attacks) to reconstruct sensitive input data, leading to potential data leakage.

In the rest of this section, we formulate a unified DL offloading formulation that integrates three subproblems – one for each objective – to determine how to place model partitions across edge and cloud resources. Then, in Sec. 4, we summarize the related works based on the optimization objectives. And, in Sec. 5, we study the interactions between performance objectives by solving our optimization problems in certain use cases introduced in related works.

As a constrained multi-objective optimization problem, we seek an offloading configuration that minimizes a weighted sum of latency, monetary cost, and privacy objectives:

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

$$s.t. \text{ constraints on inference metrics.} \tag{2}$$

Here,

- $\mathcal{L}^L$, $\mathcal{L}^C$, and $\mathcal{L}^P$ are the latency, cost, and privacy loss functions, respectively,
- w^L, w^C, w^P are nonnegative weights reflecting their relative importance.

There are other constraints, including the battery capacity of edge devices [86], developer tooling (for example, virtual machines or serverless platforms) [187] and wireless connection conditions [22], that can influence the optimization. There are also application-specific metrics, e.g., Time-to-First-Token for LLMs [74]. We focus on a subset of the optimization targets and constraints to narrow this survey. Detailed constraints for each objective are discussed in Sec. 3.1, 3.2 and 3.3.

3.1 Latency

Balancing and minimizing transmission and processing delays are essential to DL 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 first briefly introduce the latency optimization approaches. Then we present the latency formulation in Sec. 3.1.1.

We focus on a DL model composed of M partitions ($F_{pid}, pid \in 1, 2, ..., M$ in Fig. 3) 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 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 ($FLOPs_i^j(.)$). In practice, because each partition's output feeds the next, once the offloading decision for partition k changes, the same decision is typically applied to all deeper partitions to avoid extra transmission overhead [65, 81, 171].

We now overview the orthogonal approaches, such as early-exit, transmission compression, and model quantization, that an MLaaS broker can apply independently of privacy or cost-driven adaptations. The related works are detailed in Sec. 4 and the interactions among latency, privacy, and cost optimizations are discussed in Sec. 5.

Early-exit adaptation. A model partition can be adapted to reduce inference time (π_{pid} and π_{pid+1} on both sides of the dashed line in Fig. 3). Each partition pid can be accelerated 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 [63, 87, 93]. We denote the sum of the exit percentages for partition pid as β_{pid}. With β_{pid} of requests finishing at a shallow partition, internal classifiers reduce the running time of requests by proportionally cutting the communication cost of deeper partitions: only the remaining $(1-\beta_{pid})$ fraction of queries must transmit their activations, so the expected transmission delay for partition pid is scaled by the same $(1-\beta_{pid})$ factor in the term T^T_{pid} of Eq. (5).

Transmission and model compression. Partitions can also be adapted with transmission and model compression methods to mitigate both transmission and computation overhead. 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 (e.g., in Fig. 3, an Auto-Encoder Neural Network consists of dark blue layers representing an encoder and dark orange layers representing the decoder), and (2) κ_{pid} for model compression, including model size reduction approaches like knowledge distillation, neuron pruning and quantization [64, 95, 117, 132, 159, 197] as exemplified by light blue and light orange layers in Fig. 3.

Transmission compression. For partition pid, the en–decoder $\langle \mathcal{E}_{pid}, \mathcal{D}_{pid} \rangle$ squeeze x_{pid} through a latent space

$$\tilde{x}_{pid} = \mathcal{E}_{pid}\big(x_{pid}\big), \qquad |\tilde{x}_{pid}| = (1-\gamma_{pid})\,|x_{pid}|,\; 0 \le \gamma_{pid} \le 1,$$

so that only a $(1-\gamma_{pid})$ fraction of the original bytes is sent over the network. After arrival, the decoder reconstructs the activation map,

$$\hat{x}_{pid} = \mathcal{D}_{pid}\big(\tilde{x}_{pid}\big),$$

before the forward pass resumes. Because only the remaining $(1-\beta_{pid})$ requests continue beyond partition pid, the expected transmission delay for this segment is modulated by the product $(1-\beta_{pid})(1-\gamma_{pid})$ in the term T^T_{pid} of constraint (5) in the formulation.

Model compression. Model size reduction approaches like knowledge distillation, neuron pruning and quantization shrink the computation and output footprint of F_{pid}:

- *Distillation* trains a student model $\widetilde{F}_{pid}$ that mimics F_{pid} with fewer parameters, reducing FLOPs and activation size to $(1-\kappa_{pid})$ of the original.
- *Pruning* filters redundant neurons, leading to a $(1-\kappa_{pid})$ reduction in both T^C_{pid} and tensor passed to the next hop.
- *Quantization* stores weights and activations in low-bit-width integers. We fold its effect into the same factor κ_{pid} for brevity. An 8-bit model, for instance, halves memory traffic and doubles the effective SIMD throughput on hardware that packs two 8-bit multiply-accumulate (MAC) operations into the slot of a single 16-bit MAC, thereby improving parallelization and further shortening T^C_{pid}.

Together, these techniques scale the compute term T^C_{pid} in constraint (7) and the downstream transmission term T^T_{pid} by the multiplicative factor $(1-\kappa_{pid})$ as shown in Eq. (5).

3.1.1 Latency Formulation

We formulate a constrained multi-objective optimization that jointly minimizes processing and transmission delays subject to an accuracy constraint.

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

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

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

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

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

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

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

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

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

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

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 and pruning
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 requests exiting at classifier c in partition pid
β_{pid}	Percentage of requests exiting at partition pid
x_0	Source data
Q_{pid}	Number of classifiers 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 for activations and states
T^C_{pid}	Estimated computation time
$FLOPs^j_i(.)$	FLOPs from partition i to j inclusive

Table 6: Latency Optimization Formulation Notations

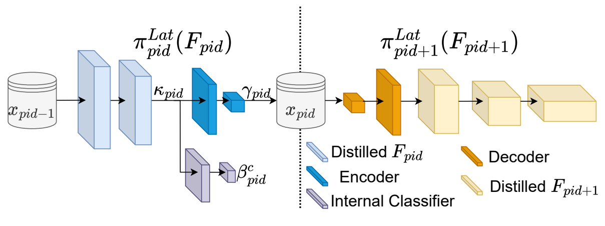

Figure 3: Illustration of latency optimization problem.

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.

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. These model adaptations include internal classifier(s) and transmission and model compression. To quantify their effect on inference latency, line 5 estimates the transmission delay from the adapted partition pid to $pid+1$ based on the input size before applying any adaptations, $Size(F_{pid}(x_{pid-1}))$, together with adaptation-related terms such as the early exiting ratio β_{pid} and the two compression ratios (γ_{pid} and κ_{pid}). Then, line 6 estimates the transmission time required to send the source data to the location of the first NN partition, where γ_0 denotes the compression ratio applied to the source data.

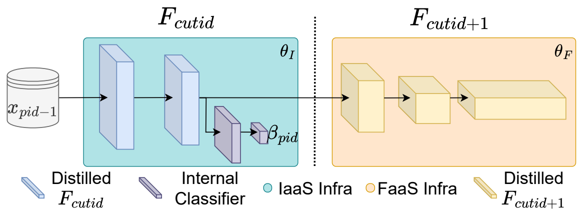

Figure 4: Illustration of cost($) optimization problem.

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
θ_I	VM capacity
θ_F	Function capacity

Table 7: Cost($) Optimization Formulation Notations

In line 7, $FLOPs^{pid}_{pid}(x_{pid-1}, \pi_{pid}, F_{pid})$ denotes the profiled FLOPs (FLoating-point OPerations) for partition pid (F_{pid}), given its input (x_{pid-1}) and model adaptation (π_{pid}). The subscript and superscript indicate the start and end partitions included in the count. for a single partition, these indices are the same. Although adapted-model FLOPs can be computed analytically, actual computation time T^C_{pid} also depends on system-level factors, including scheduling, virtualization, etc., which lie outside the scope of this survey. We therefore adopt a profiling-based formulation and treat the execution environment as a black box.

3.2 Monetary Cost

In this section, we categorize cost optimization approaches and introduce our monetary cost formulation (Sec. 3.2.1). Resource provisioning approaches based on resource cost ($) for DL 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 sparsely activated ML models where each portion faces a varying workload, fine-grained resource provisioning and load balancing for submodels are essential to achieve cost-efficient ML inference.

In our formulation, we minimize the combined costs of provisioning Infrastructure-as-a-Service (IaaS) and Function-as-a-Service (FaaS) platforms, regardless of whether they are deployed at the edge or in the cloud. FaaS charges users only for the actual execution time of the deployed model, making it particularly cost-efficient for low-rate or bursty workloads – for example, holiday traffic spikes in DNN-based vehicle localization [5, 67, 71, 158]. Conversely, IaaS platforms automatically scale virtual machines (VMs), which incur longer cold-start delays (for hardware and OS provisioning) and are billed for the entire time the resources are reserved. Because VM scaling is coarse-grained, users typically provision based on service-level objectives (SLOs) rather than real-time workload fluctuations, often leading to over-provisioning [143]. However, when a workload maintains high utilization, characterized, for instance, by a steady ingestion rate around the mean $\pm$ one standard deviation under a Poisson model [77, 143], IaaS can be more economical than serverless functions.

3.2.1 Monetary Cost Formulation

Motivated by recent advances in sparsely activated models [61] and the inability of edge devices to host large neural networks [81], we represent a sequential DNN with internal classifiers as a dependent acyclic graph of submodels (Fig. 4). For inference, partition F_{pid} produces an early exit for a fraction β_{pid} of requests with its internal classifier. The remaining $(1-\beta_{pid})$ portion continues to partition F_{pid+1}. Shallow partitions thus face a steady, high-rate workload, whereas deeper partitions see a lower request rate. Consequently, for a constant arrival rate of N requests/s, we provision VMs to handle up to $r_{\max}$ of those requests at the shallow partitions, ensuring high VM utilization, while routing the remaining requests to FaaS to avoid under-utilizing any individual VM.

$$\begin{aligned}
\mathcal{L}^C = \min_{cutid,\theta_F,\theta_I,\alpha^k_{pid}} \; & C_I(\theta_I)T_I \sum_{pid=1}^{cutid} \beta_{pid} \\
& + C_F(\theta_F)T_F \sum_{pid=cutid+1}^{M} \beta_{pid} && (1) \\
s.t. \quad & Latency \geq T_I + T_F && (2) \\
& \sum_{pid=1}^{M} \sum_{c=1}^{Q_{pid}} \beta^c_{pid} A^c_{\pi_{pid}} \geq A_{tar} && (3) \\
\beta_{pid} &= \sum_{c=1}^{Q_{pid}} Pr(\alpha'^c_{pid} > \alpha^c_{pid}) \\
&= \sum_{c=1}^{Q_{pid}} \beta^c_{pid} && (4) \\
T_F &= \frac{FLOPs^M_{cutid+1}(x_{cutid})}{\theta_F} + T^{cutid}_{cold} && (5) \\
T_I &= \frac{FLOPs^{cutid}_1(x_0)}{\theta_I} + T^{cutid}_{trans} && (6) \\
& cutid \in [1, M],\ \alpha^k_{pid} \in [0,1], && (7) \\
& \theta_F \in \{\text{FaaS Capacities}\}, && (8) \\
& \theta_I \in \{\text{IaaS Capacities}\} && (9)
\end{aligned}$$

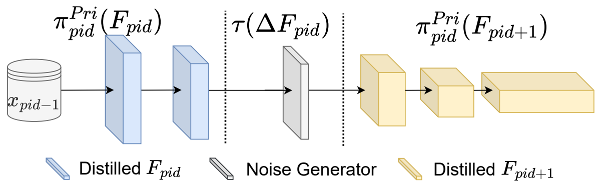

Figure 5: Illustration of privacy optimization problem.

The above optimization targets a steady arrival rate of $r_{\max}$ DNN inference requests per second, enough to fully utilize the VMs. It chooses between IaaS, FaaS, or *hybrid offloading* – offloading some requests from IaaS to FaaS to complete their deep-layer processing. The decision variables are the FaaS configuration (θ_F), the IaaS configuration (θ_I), the internal classifier thresholds (α^c_{pid}), and the partition index ($cutid$, assuming two partitions), all subject to the latency constraint in line 2.

C_I and C_F map resource configurations to monetary cost. Then, we estimate the running cost, based on the average durations T_I (VM reservation time) and T_F (FaaS execution time) profiled for each forward pass. In line 4, β_{pid} is the fraction of requests exiting at partition pid, determined by confidence thresholds α^c_{pid}. Lines 5 and 6 define the profiled mean durations for FaaS and IaaS, respectively. For a given $cutid$, motivated by previous work [82], the duration is computed by dividing the required FLOPs ($FLOPs^M_{cutid+1}$ from partition $cutid+1$ to M) by the provisioned capacity (θ_F or θ_I), assuming once offloaded to FaaS, execution does not revert to IaaS. This may overestimate utilization since some requests exit early. To account for early exits, we weigh T_F by $\sum_{pid=cutid+1}^{M} \beta_{pid}$ and T_I by $\sum_{pid=1}^{cutid} \beta_{pid}$ in Eq. (1). We also include the hidden-variable transmission delay T^{cutid}_{trans} in T_I (line 6), since the serverless function has not yet been invoked and thus does not incur FaaS billing until execution. Conversely, the serverless cold-start delay T^{cutid}_{cold} is included in T_F (line 5), as it involves model loading and hardware setup that are billable.[2]

Next, given the VM configuration for $r_{\max}$ requests/s, we optimize load-balancing so that $r_{\max}$ requests are served by shallow partitions on fully utilized VMs, with the remainder directed to FaaS. Note that the sparsity of early-exit models is highly dependent on the input-data distribution and must be profiled offline. When the probability of FaaS provisioning increases, resulting in higher costs, we must update our workload distribution profile and resource configurations.

3.3 Privacy

We study the privacy of source data in distributed DNN inference applications. This section overviews the concepts – accessible by an MLaaS broker – for defense against model inversion attacks (MIA) [42, 180, 196]. We formulate the privacy optimization problem in Sec. 3.3.3. Many of the techniques discussed are applied during model training, rather than only at inference Practical countermeasures for MIA are presented in detail in Section 4.3.

3.3.1 Threat Model: Model Inversion Attack (MIA)

We consider an *honest-but-curious* adversary who observes the hidden variables x_{pid} transmitted between two partitions of a distributed DNN. The adversary stores these activations and trains an auto-encoder (or any other reconstructor) $\mathcal{R}_{\text{MIA}}$ *offline* with public data, then outputs $\hat{x}_0 = \mathcal{R}_{\text{MIA}}(x_{pid})$ at inference time. We measure privacy leakage by the expected reconstruction error, e.g., mean square error (MSE),

$$\text{Leak}_{pid} = \mathop{\mathbb{E}}_{x_0 \sim \mathcal{D}} \left[\|x_0 - \hat{x}_0\|_2^2 \right],$$

where $\mathcal{D}$ is the distribution of user inputs. A *smaller* MSE implies a *stronger* attack. Such attacks have been well explored in the research community [38, 45, 96, 193] and in particular the Prompt Inversion Attack (PIA) for LLMs [113, 139].

3.3.2 Privacy-oriented Adaptation

To mitigate MIA, an MLaaS broker can leverage two orthogonal defenses – **regularization** and **perturbation** – applied *independently* of any latency or monetary cost-driven optimizations:

1. *Regularization.* We fine-tune each partition with a privacy-aware objective, denoted by π^{pri}_{pid}, that explicitly penalizes the attacker's reconstruction loss [58, 96, 175, 210].

2. *Perturbation.* We add a noise layer $\tau(\Delta F_{pid})$ based on the output sensitivity ΔF_{pid} and bounded by a scalar λ [58, 126].

Figure 5 illustrates the three stages (separated by dashed lines): partition pid (left), the noise layer $\tau(\cdot)$ (center), and partition $pid+1$ (right). However, we note that the effectiveness of such remedies in handling PIA for LLMs remains under-explored due to the prohibitive model training overhead. We next formulate the privacy optimization problem.

[2]Major FaaS providers keep containers warm to minimize cold starts [18, 48, 123, 158], so this term may overestimate FaaS cost. VM cold starts are omitted since we assume long-running VMs.

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 F_{pid}'s output
w_{CE}, w_p	Weights for loss terms
Thr_{CE}	Maximum allowed cross-entropy loss
$F^{-1}_{pid}(\cdot)$	Learned inverse (attacker's surrogate)

Table 8: Privacy Optimization Formulation Notations

3.3.3 Privacy Formulation

With the notations in Table 8, we formulate the *privacy* objective:

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

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

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

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

Equation (1) *maximizes* the attacker's error ($-$sign) while keeping model accuracy above a threshold (2). In this formulation, $CE(\hat{y},y)$ denotes the cross-entropy of the model predictions, and $MSE(F^{-1}_{pid}(x_{pid}), x_{pid-1})$ quantifies the fidelity of the attacker's reconstruction. The forward rule (3) shows how bounding (λ) and random noise $\tau(\cdot)$ are injected between partitions. The interaction with latency and monetary cost is discussed in Sec. 5; here we isolate privacy.

4 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 monetary cost ($). In this section, we detail the solution techniques to the optimization sub-objectives, such as early exits, compression, and privacy-preserving inference approaches. Based on the existing works, we then discuss the interactions among sub-objectives in Sec. 5. Furthermore, we identify open issues in Sec. 6. Overall, these solution techniques can serve as valuable control mechanisms for ML service providers, improving Quality of Service (QoS), and increasing revenue.

4.1 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 [81, 194]. Then, we discuss internal classifiers which allow early exit and save computation for deep layers [36, 87, 92–94]. Next, we examine transmission data and model compression approaches [40, 63, 87, 117, 132, 159, 207, 212].

4.1.1 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 [55, 166]. Thus, delays, especially inference durations composed of transmission and processing delay, T_{pid}^T and T_{pid}^C respectively, can be modeled using regression methods, by profiling the NN across cloud and resource-limited edge environments [81, 194]. Prior work estimates transmission and processing delays for various model configurations, factoring in compute resources and input sizes to devise deployment plans for M NN partitions that minimize latency and energy consumption [81].

Our latency formulation in Sec. 3.1.1 captures the total delay, consisting of transmission and processing delays, for a model with M partitions ($\xi_0^T T_0^T + \sum_{pid=1}^{M} T_{pid}$ in line 1). However, feasible solutions may not always exist for a given model architecture or environment, particularly when resource availability is constrained. Next, we explore orthogonal methods to reduce demands on transmission and processing resources.

4.1.2 Early Exits

Background. Deep layers of a DNN model tend to focus on fine details, which, however, can result in misclassifications. While, shallow layers extract high-level features, which can be sufficient for accurate request classification. Such observation is described as *overthinking* [82]. Related research [82, 188, 214] addresses this concern, proposing the reuse of features extracted from various layers to improve the inference latency through *internal classifiers* [170],

These classifiers share a structure similar to traditional NN classifiers, comprising feature reduction (pooling) layers, fully connected layers, and an activation function. Except, internal classifiers are attached to the hidden layers. To trigger early exits, one can configure a threshold [82] for the Bayesian probabilities of class predictions at each classifier [53, 150].

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 inference, leading to improved accuracy and latency. Early exits prevent excessive forwarding of requests (hidden variables) to deep layers for classification. As exemplified in Fig. 6, an internal classifier allows β_{pid} portion of requests to exit the model partition F_{pid}, highlighted in blue.

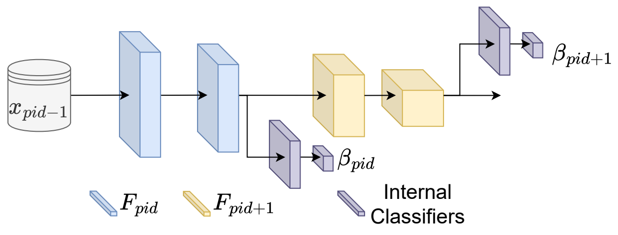

Figure 6: Internal Classifier Architecture: Each internal classifier allows requests to exit in the middle of an NN. For example, β_{pid} of requests exit at NN partition F_{pid}.

Methods. Recent research [63, 87, 171] models the relationship between the confidence threshold (α) of internal classifiers and the proportion of early exits (β) when formulating inference latency. Our latency optimization framework incorporate such relationship, allowing β to be tuned via α to meet a specific mean latency target (lines 2 and 3 in Sec. 3.1.1.)

Lowering the confidence threshold (α) during inference can also negatively impact the inference accuracy ($\sum_{pid=1}^{M} \sum_{c=1}^{Q_{pid}} \beta_{pid}^c * A_{\pi_{pid}}^c$, line 2 in our latency formulation). To maintain high accuracy, previous studies explore multi-objective optimization including confidence threshold and dynamic layers offloading between the edge and cloud based on network conditions, as characterized by lines 4, 5, and 7. For example, SPINN [87] 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, which allows high confidence threshold and high accuracy. 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, without significantly compromising accuracy.

4.1.3 Input and output compression

Background. During inference, not all features are necessary for a classification task. Apart from traditional statistical or heuristic methods [177], Deep Neural Networks (DNNs), specifically Auto-Encoder NNs, can facilitate feature selection to preserve prediction performance [159]. An Auto-Encoder NN consists of two components: an encoder, which transforms inputs to a compressed representation, and a decoder, responsible for inverting the dimensionality reduction [152]. On the other hand, model compression methods can also reduce feature size. These methods will be explored further in Sec. 4.1.4.

Methods. In our latency optimization ($\mathcal{L}^L$ in Sec. 3.1.1), we denote the cropping and compression of input data with rate γ_0. 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 compressed image inputs, can help reduce feature dimension at the cost of accuracy. In particular, since certain activation functions, for example relu [3], produce zero or near-zero outputs, compression from a dense matrix into a sparse matrix is both storage and transmission efficient [63]. Moreover, related works [63, 68, 111, 156] explore the quantization of weights and intermediate data representations. Rather than using double precision floats, these methods consider 8-bit [68] or in the extreme case single-bit [111] approximations.

Several prior works investigate content-aware transmission compression. 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, thereby reducing bandwidth and latency [151]. Likewise, RoI extraction in multimodal LLM pipelines enables processing of high-resolution imagery by isolating the most informative patches for downstream analysis [25, 208]. For text-based LLMs, prompt-compression techniques prune tokens and sentences that are considered uninformative with respect to the query, improving responsiveness without necessarily sacrificing relevance [74–76, 98, 99, 104, 135, 201, 204–206]. Although these approaches generally reduce transmission delay, their impact on accuracy depends on the quality of the cropping or pruning mechanism. Another study further realizes that concentrating on relevant data can both cut transmission costs and improve predictive performance [132].

Previous work has also applied ML-based dimensionality reduction to reduce transmission data and maintain accuracy. One idea is to insert a *bottleneck* between two neural network partitions using an Auto-Encoder NN [40, 65, 117, 159, 197, 212]. The Auto-Encoder is trained by minimizing the Mean Squared Error (MSE) loss between the input and output data (the reconstruction). In this setup, the encoder projects 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 leads to a compact representation of intermediate data, enhancing efficiency minimal accuracy degradation.

For the attention mechanism, low-rank approximation (LoRA) has been applied to compress sparse attention matrices [165]. Because attention matrices scale quadratically with sequence length, approximating them via low-rank factors significantly lowers both computational and memory costs.

Other approaches jointly optimize the backbone and an intermediate autoencoder (AE) to better preserve end-to-end accuracy [65, 159, 197]. By training the AE together with the task model, the compressed latent space tends to retain features that are most relevant for the downstream prediction task. 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 [64] uses explainable AI techniques, including Integrated Gradients [168], 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. Input-dependent techniques, including JPEG and RoI-based input pruning for image or LLM prompts, set the compression rate of model input (γ_0). Intermediate-data-dependent techniques, including ML-based dimension reduction and pruning using Auto-Encoder or LoRA, as well as quantization, set the compression rate of intermediate data (γ_{pid}). Meanwhile, the data compression techniques (π_{pid} in line 4) also add FLOPs in each partition ($FLOPs_{pid}^{pid}(x_{pid-1}, \pi_{pid}, F_{pid})$ in line 7), creating a trade-off among reduced transmission overhead (γ_{pid}), increased computation overhead ($FLOPs_{pid}^{pid}(x_{pid-1}, \pi_{pid}, F_{pid})$, and potentially compromised model accuracy ($\sum_{pid=1}^{M}\sum_{c=1}^{Q_{pid}} \beta_{pid}^{c} * A_{\pi_{pid}}^{c}$ in line 2), which a system architect must evaluate when choosing compression methods. We summarize the data compression methods 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 handle the discrete space [68, 129].

4.1.4 Model Compression and Knowledge Distillation

Background. Deep Learning (DL) models can be customized to reduce inference latency by simplifying neural architectures while maintaining acceptable accuracy. Common categories include: (i) *quantization* [68, 100, 115], which lowers numerical precision; (ii) *pruning and sparsification* [106, 195], which edit or reduce the data flow across neural network layers; (iii) *dynamic inference* techniques [57, 181, 184, 191], which adaptively skip layers or channels to save computation; and (iv) *knowledge distillation* [50, 59], which trains lightweight student models to mimic larger teacher models.

For large models such as LLMs, additional compute-reduction strategies include adjusting quantization parameters via static analysis (e.g., AWQ [101], SmoothQuant [189],

Activation Compression Method	**Pre-Processing** [132]	**Heuristic** [63]	**Quantization** [63, 68, 111, 156]	**AE** [40, 65, 117, 159, 197, 212]	**AE(XAI)** [64, 65, 117, 159, 197]
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 [132] 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).

OPTQ [41]), sparse-activation processing (e.g., EIE [56]), mixture-of-experts architectures [51, 160], and speculative decoding [23, 91, 121], where smaller draft models generate tokens for verification by a larger model. Hybrid inference strategies also offload portions of generation to simplified on-device models while refining outputs using full-scale cloud models [32, 122]. These approaches share a common goal of lowering inference latency or resource usage, though with varying impacts on accuracy. In some cases, simplification can even improve generalization by reducing shortcut learning and mitigating overfitting [46, 69]. In this work, we focus on how these compression principles integrate with partitioned neural networks executed across edge–cloud environments.

Methods. We use π_{pid} to denote model adaptations applied to partition *pid*, including quantization, pruning, sparsification, dynamic inference, and distillation. With an adaptation π_{pid}, $FLOPs_{pid}^{pid}(x_{pid-1}, \pi_{pid}, F_{pid})$ estimates the computation cost of that partition. Compression reduces (i) the compute time and (ii) the size of hidden representations transmitted to the next device, thereby improving κ_{pid} and γ_{pid} (cf. Sec. 4.1.3).

Quantization methods such as PTQ and QAT [65, 95, 129] reduce precision for weights and activations, directly lowering arithmetic cost. Sparsification and pruning techniques, including CLIO [63] and BBNet [212], reduce the number of nonzero computations and thus $FLOPs_{pid}^{pid}$. Dynamic inference approaches [57, 181, 184, 191] further decrease FLOPs by adaptively skipping layers or channels based on input-dependent conditions. These techniques all reduce T_{pid}^{C} but may degrade accuracy under aggressive compression.

For highly resource-constrained edge partitions, aggressive compression may significantly distort features. Lee et al. [90] propose using a deep decoder at the cloud node to reconstruct high-dimensional representations from aggressively compressed activations. This approach allows more substantial edge-side compression while maintaining end-to-end accuracy.

Instead of compressing an existing submodel, a compact student model can replace an entire partition. Partition-wise distillation [117] trains each student partition using logits or intermediate features supplied by a teacher network, yielding low-latency components with minimal accuracy loss. Self-distillation variants [202, 207, 211] can further regularize intermediate classifiers and reduce overfitting by propagating soft labels across different depths of the model.

Summary of model compression methods. In our latency optimization framework (Sec. 3.1), π_{pid} unifies these adaptations and determines each partition's compute time:

$$T_{pid}^{C} = \frac{FLOPs_{pid}^{pid}(x_{pid-1}, \pi_{pid}, F_{pid})}{\mu_{pid}}.$$

Quantization, pruning, sparsification, dynamic inference, and distillation are complementary and should be considered jointly when optimizing partitioned model complexity and placement. Table 10 summarizes the trade-offs among these compression techniques.

Among these techniques, knowledge distillation offers the greatest practicality, as student model architectures and distillation objectives can be extensively customized to preserve accuracy under tight resource budgets. However, this flexibility comes with higher computational overhead during training.

In contrast, quantization provides more limited configurability but incurs the lowest computation overhead and is widely adopted in practice for latency-sensitive deployments. The effectiveness of weight pruning varies with the underlying data distribution. For instance, pruning weights close to zero is a well-established heuristic for maintaining accuracy while reducing computation and data size, but mask-based pruning often requires data-specific training, leveraging Auto Encoder or Explainable AI tools, to construct effective sparsity patterns [31]. As a result, pruning methods generally fall between distillation and quantization in terms of practicality and computational overhead.

4.2 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 [47]. Previous work discusses splitting model states [141], kernel fusion techniques [27], and sparse attention mechanisms [199] 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 $\$700,000$ [136]. 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.

4.2.1 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 [145] 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 generic edge and cloud resources [63, 87, 107, 137, 213]. Later work also considers specific cost models according to the cloud services, such as Infrastructure-as-a-Service (IaaS) and Function-as-a-Service (FaaS) [71, 143], to balance processing and transmission demands while minimizing expenses. However, detailed cost analyses using real-world cloud resources for low-cost ($) ML serving remain limited. Many studies model resource expenses on the edge and in the cloud using generic unit costs [107, 137, 213]. However, the specific provisioning factors for each edge and cloud service, including container cold starts and billing time granularity, are critical to minimizing real-world cloud usage costs.

Summary of cost-saving methods: Based on our cost formulation (Sec. 3.2.1), the accumulated running time of heterogeneous resources introduces heterogeneous resource-usage costs. From the resource-orchestration perspective, the various pricing components of provisioning services, including cold-start time (T_{cold}^{cutid}), transmission time (T_{trans}^{cutid}), and processing time ($\frac{FLOPs_{cutid+1}^{M}(x_{cutid})}{\theta_F}$ and $\frac{FLOPs_{1}^{cutid}(x_0)}{\theta_I}$), significantly affect provisioning decisions. We summarize related work in Table 11. Overall, coarse-grained cost analyses that abstract away application- or model-specific details are more generalizable. While fine-grained methods can better approach cost-optimal solutions, they often rely on specialized model features, such as internal classifiers.

4.3 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 2.1.3, an adversary could reconstruct the intermediate data transmitted between DNN partitions [175] using an Auto-Encoder Neural Network. To protect against this vulnerability for model inference, previous research adds *Perturbation* to intermediate data [58, 126] or incorporates a *Regularization* step during training [58, 96, 175, 210]. Such methods preserve only essential features for ML tasks and remove sensitive information.

4.3.1 Perturbation

Background. Differential privacy (DP) improves privacy in statistical databases by adding noise to query outputs proportional to the *sensitivity* of the query [35, 203]. Consider a query $f : D \rightarrow R$ on a dataset D with samples $x, x^{'} \in D$. The global sensitivity Δf of this query is defined as:

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

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

Model Compression Method	**Quantization** [65,95]	**KD** [117]	**Weight-Pruning** [63,90,212]
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 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.

Analysis Granularity	**Coarse** [63,87,107,137,213]	**Fine-Grained** [71,143]
Cost Saving Optimality	Low	High
Model Structure Flexibility	High	Low

Table 11: A comparison of cost analysis methodologies for ML systems. *Coarse-Grained Analysis* treats the ML model as a monolith. This approach offers high *Model Structure Flexibility* but overlooks potential savings by not optimizing resources for individual components, leading to lower *Cost Saving Optimality*. Conversely, *Fine-Grained Analysis* models the cost of individual layers or partitions. This allows for highly optimized resource allocation (e.g., VMs vs. FaaS for different parts of a model), but is less flexible and requires a more detailed system model.

the sensitivity in model partition output [1,203].

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

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 [1].

Previous studies apply this practical DP implementation in DP-SGD [1,200] during training to mitigate the risk of reconstructing training datasets from the served 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. This approach complicates membership inference attacks, in which adversaries deduce whether a sample was part of the training data based on prediction logits [43].

In edge inference settings, recent work suggests injecting noise to hidden variables that obscure sensitive information, for example, race, age, or gender, transmitted over the Internet [58, 126].

Methods. In our privacy optimization formulation (Sec. 3.3), 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 [58, 116, 126, 203] 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 at model serving time. Meanwhile, the model retains its capacity to extract relevant information for accurate predictions.

4.3.2 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 with 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. 3.3), 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 [96,175,210] and model partition placements [58,210].

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 [175]. Alternatively, additional training epochs can be dedicated to optimizing the privacy objective [210]. For more task-specific solutions, ResSFL [96] 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.

Summary of privacy-preserving methods: As our privacy formulation (Sec. 3.3.3) shows, prior work mitigates model-inversion attacks (MIA) and prompt-inversion attacks (PIA) by reducing the amount of recoverable information encoded in shallow layers. In general, existing solutions introduce either perturbation layers ($\tau(\Delta f)$ in line 3 of our privacy formulation in Sec. 3.3.3) during inference or regularization terms during training ($MSE(F_{pid}^{-1}(x_{pid}), x_{pid-1})$ in line 1 of our formulation in Sec. 3.3.3). We summarize these approaches in Table 12. Regularization-based methods generally require end-to-end model training and tend to preserve accuracy more effectively, but they incur higher training overhead. In contrast, perturbation-based methods are lightweight and can often be applied post hoc, though they usually offer weaker guarantees in either accuracy or privacy. Finally, although these techniques have been extensively evaluated for vision models and compact edge networks, their applicability to large language models remains insufficiently explored. Adapting them to the scale, tokenized structure, and high-dimensional representations of modern LLMs is an open research challenge.

5 Multi-Objective Optimization Case Studies

Optimizing for latency ($\mathcal{L}^L$), monetary cost ($\mathcal{L}^C$), and data privacy ($\mathcal{L}^P$) simultaneously presents a significant challenge, as these objectives often conflict. An improvement in one area can adversely affect another. For instance, prior work on CIFAR-10 has shown that reducing the edge model's depth from 7 to 4 layers – a change that could decrease latency – degrades privacy, causing the attacker's top-1 misclassification rate on reconstructed images to fall from approximately 80% to below 40% [210]. Similarly, techniques designed to accelerate inference, such as internal classifiers (β_{pid}) or latent compression (γ_{pid}), can lead to unpredictable resource usage. This unpredictability may result in the under-utilization of provisioned virtual machines, thereby increasing the effective monetary cost per request.

However, these same mechanisms can also be used to co-optimize multiple objectives when applied in the right context. For example, early exits reduce traffic to deeper model layers, allowing a small set of always-on VMs to handle the "hot path" (frequent, low-latency requests) while offloading less frequent "cold" traffic to serverless functions (FaaS). This approach can simultaneously reduce both latency and cost. In this section, we therefore explore the tradeoffs inherent in our formulation through a series of use cases, analyzing scenarios where a subset of objectives is relaxed to maximize performance on others.

5.1 Latency & Cost

In this section, we analyze scenarios where data privacy is a relaxed constraint, allowing us to focus on the trade-off between inference latency and monetary cost. This situation is common for large models like LLM chatbots (e.g., ChatGPT [133] and Gemini [169]), which are too computationally intensive to be deployed on user devices and must run on remote servers. Accordingly, we now examine how our formulation can be used to balance these two objectives.

Our optimization of latency ($\mathcal{L}^L$) is guided by the objective in line 1, which minimizes total inference time:

$$\mathcal{L}^L = \min_{pid,\alpha_{pid}^c,\kappa_{pid},\gamma_{pid}} (\xi_0^T T_0^T + \sum_{pid=1}^{M} T_{pid})$$

This total time is composed of transmission delay (T_{pid}^T), defined in line 5, and processing delay (T_{pid}^C), defined in line 7:

$$T_{pid}^T = \frac{(1-\kappa_{pid})(1-\beta_{pid})(1-\gamma_{pid})\text{Size}(F_{pid}(x_{pid-1}))}{bandwidth}$$
$$T_{pid}^C = \frac{FLOPs_{pid}^{pid}(x_{pid-1},\pi_{pid},F_{pid})}{\mu_{pid}}$$

Similarly, our optimization of monetary cost ($\mathcal{L}^C$) is guided by the objective in line 1. This function models the combined cost of IaaS resources up to a partition point, `cutid`, and FaaS resources thereafter:

$$\mathcal{L}^C = \min_{cutid,\theta_F,\theta_I,\alpha_{pid}^k} C_I(\theta_I)T_I \sum_{pid=1}^{cutid} \beta_{pid} + C_F(\theta_F)T_F \sum_{pid=cutid+1}^{M} \beta_{pid}$$

This cost is weighted by the probability that a request will reach a given layer, β_{pid}, which is derived from the workload distribution as shown in line 4:

$$\beta_{pid} = \sum_{c=1}^{Q_{pid}} \Pr(\alpha_{pid}^{\prime c} > \alpha_{pid}^c) = \sum_{c=1}^{Q_{pid}} \beta_{pid}^c$$

Although latency and cost are coupled, reducing one does not automatically reduce the other. Techniques that lower latency, such as data compression (κ_{pid}, γ_{pid}) and early exits (β_{pid}), directly influence the total execution time (T_I, T_F), which in turn determines monetary cost. However, this relationship is complex. For example, while IaaS platforms may offer more cost-efficient hardware than FaaS, dedicating a full VM to a model is not always cheaper. In a DNN with internal classifiers, many requests may exit early, leading to short, unpredictable inference times [82, 87]. And in a hybrid LLM setup where easy queries are handled by a local LLM while complex one needs more sophisticated cloud LLM [30]. In particular, in Hybrid LLM, with 1% drop in

Defense Method	Perturbation [58, 116, 126, 203]	Regularization [58, 96, 175, 210]
Training Overhead	Medium	High
Required Model Retraining	Partial or Complete	Complete
Influence on Accuracy	High	Medium

Table 12: A comparison of two common defenses against model-inversion attacks. *Perturbation* methods add noise to intermediate data, while *Regularization* methods add a privacy-penalty term to the training loss. *Perturbation* often has lower *training overhead* because noise-generating components can sometimes be trained separately [126]. However, this training paradigm can lead to a significant accuracy drop (e.g., differential privacy vs. CPA-DC in Federated Split Learning [210]). In contrast, *Regularization* requires end-to-end retraining, which incurs higher overhead but typically affects accuracy less. Although these techniques have been studied for traditional MIA, their substantial training cost makes them largely impractical for defending against *prompt-inversion attacks* (PIA) in large-scale LLMs.

BART score of responses (i.e., use smaller edge LLM model), they reduced 22% traffic to the cloud LLM (i.e., api calls to GPT-3.5-turbo) [30]. These short jobs can under-utilize a provisioned IaaS VM, making it less cost-efficient. In contrast, a FaaS platform, which bills based on precise execution time, is often more cost-effective for these variable, short-running requests.

To jointly optimize inference latency and monetary cost, prior research has explored architectural-aware resource provisioning and model offloading. One prominent approach involves **hybrid edge-cloud systems**, which route computationally simple queries to local edge devices while offloading complex requests to more powerful, highly-parameterized models (e.g., Large Language Models) in the cloud [30, 157]. Other works leverage **internal classifiers** and ensemble model architectures, allowing for on demand fine-grained resource provisioning for layers deployed on the edge or in the cloud [61, 83, 87, 138, 172, 209].

When considering resource costs rather than per-call API fees for services or models, the unique pricing models of cloud providers also affect ML resource-provisioning decisions. In LIBRA [143], the authors 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. They achieved 20% cost reduction by load balancing. 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 [52]. 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 [158].

For Large Language Models (LLMs), where output lengths are highly unpredictable, recent work has focused on the unique characteristics of the auto-regressive decoding phase. Specifically, because the Key-Value (KV) cache of previous tokens remains static during the generation of a new token, it is possible to migrate the KV cache to a different node with sufficient GPU memory [167]. This technique enables **low-downtime migration** of ongoing LLM inference jobs, which is critical for maintaining low 99th-percentile latency. Furthermore, this low-overhead migration method facilitates dynamic resource allocation, allowing for minimal over-provisioning and thereby reducing monetary costs.

5.2 Latency & Privacy

In our second scenario, we examine privacy-sensitive applications where minimizing inference latency is critical, while monetary cost is a less stringent constraint. Such applications are common in environments like smart hospitals [19] or in enterprise use cases for LLM chatbots, where prompts may contain proprietary information such as company names or contact details [113, 139].

In these contexts, strict data privacy requirements often mandate that source data cannot leave the user's local facility. Despite this on-premise processing constraint, the machine learning system is still required to deliver high-quality outputs with low inference latency, creating a significant technical challenge.

As defined in Sec. 3.3.1, our privacy formulation centers on a **Model Inversion Attack (MIA)**. We consider an environment where a DNN is partitioned across a directed acyclic graph, and the threat goal is to infer the source data from the intermediate activations exchanged between partitions.

In this threat model, an honest-but-curious adversary trains a **reconstructor model** ($\mathcal{R}_{\text{MIA}}$) that takes an intermediate activation, x_{pid}, as input and attempts to generate a reconstruction of the original source data, $\hat{x}_0 = \mathcal{R}_{\text{MIA}}(x_{pid})$. The resulting privacy leakage from any partition `pid` can be quantified by the Mean Squared Error (MSE) between the original data and its reconstruction:

$$\text{Leak}_{pid} = \mathop{\mathbb{E}}_{x_0 \sim \mathcal{D}} \left[\|x_0 - \hat{x}_0\|_2^2 \right]$$

Accordingly, our privacy-optimization objective, introduced in line 1 of privacy formulation (Sec. 3.3.3), is formulated as:

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

This objective function seeks to strike a balance between two competing goals: maintaining model accuracy, measured by

Cross-Entropy (CE), and preserving source data privacy by minimizing the invertibility of intermediate representations, measured by MSE.

The privacy-enhancing techniques we consider alter the intermediate data, x_{pid}, exchanged between partitions, as shown in line 3:

$$x_{pid} = \lambda\pi_{pid}^{pri}(F_{pid})(x_{pid-1}) + \tau(\Delta F_{pid})$$

These alterations, introduced by the privacy-aware training approaches ($\pi_{pid}^{pri}(\cdot)$ and $\tau(\cdot)$), directly impact the transmission and processing delays (T_{pid}^{T} and T_{pid}^{C}) in our latency formulation. The function $\pi_{pid}^{pri}(\cdot)$ is analogous to the $\pi_{pid}(\cdot)$ function used for latency optimization (lines 4 and 7), as both represent paradigms including model compression, data compression, and partition offloading.

For traditional DNNs used in tasks like image classification (e.g., VGG), the overhead from these privacy measures can be minimal. Techniques such as regularization [58, 96, 210] or data perturbation [58, 126] often introduce few architectural changes during inference. For example, ResSFL [96] introduce an inversion network of roughly 76.2,M FLOPs together with a client model of 21.5,M FLOPs during model training, while maintaining an MSE of 0.02 between source and reconstructed data during inference. Furthermore, any resulting increase in processing or transmission overhead can frequently be compensated for by applying model compression techniques [110, 197].

In the context of modern Large Language Models (LLMs), the threat of model inversion has evolved into the **Prompt Inversion Attack (PIA)**. The motivation for such attacks is strong: user prompts can contain sensitive personal information, while system prompts often represent valuable intellectual property. To mitigate these risks, some works propose offloading schemes that partition the LLM between the edge and the cloud, preventing the raw prompt from being transmitted over the network while also managing GPU memory constraints. However, this does not solve the core problem, as an honest-but-curious adversary can still attempt to reconstruct the prompt by capturing the intermediate activations between model partitions [113, 139].

At first glance, inverting LLMs appears more difficult than inverting traditional CNNs like VGG. LLMs are significantly deeper and more non-linear, and their inputs – discrete token embeddings – are theoretically harder to reconstruct than continuous image data. Despite these challenges, recent research has demonstrated successful prompt reconstruction by leveraging open-source base model weights and LoRA adapters [182]. Furthermore, traditional privacy-preserving techniques like regularization and data perturbation are generally impractical for models of this scale, as they introduce prohibitive training overhead and can cause significant performance degradation. Therefore, developing effective and efficient methods to guarantee the privacy of prompt inputs remains a critical and open research challenge.

5.3 Latency, Cost & Privacy

In our third and final scenario, we address the most comprehensive challenge: the simultaneous optimization of ML inference latency, source data privacy, and monetary cost. This setting is crucial for deploying practical, privacy-sensitive applications that must also operate under tight latency and budget constraints. While our previous sections analyzed pairwise trade-offs, the economic realities of modern large-scale models necessitate a holistic approach. The exponential growth in inference expenses has made per-request cost a critical metric [44, 54], meaning a truly viable solution must co-optimize all three objectives.

Achieving this is exceptionally challenging, as improvements in one area often create complex, competing effects on the others. Consider the impact of a single architectural decision designed to enhance privacy against a Model Inversion Attack (MIA). A straightforward strategy is to increase the number of layers processed locally on a user's device (the head partition) before offloading to a remote server. This strengthens privacy by applying more non-linear transformations but introduces a multifaceted trade-off:

- **Transmission Latency** (T_{pid}^{T}) may decrease, as the resulting intermediate data sent to the server is often smaller.
- **Processing Latency** (T_{pid}^{C}) on the user's device will increase due to the heavier computational load.
- **Monetary Cost** may be reduced, as less computation is required from paid server resources.

This single example illustrates the difficulty finding a global optimum across latency, privacy, and cost.

Furthermore, the choice of cloud service for each model partition is a critical factor due to differing pricing models (e.g., AWS EC2 [7] vs. AWS Lambda [9]). As established in our cost formulation (Sec. 3.2.1), we can model a hybrid system that uses a cost-efficient EC2 instance for the initial, privacy-sensitive partitions and a more expensive, serverless Lambda function for the deeper layers.

In this model, architectural decisions directly translate to monetary cost. For instance, one can shift more layers of computation onto the EC2 instance (as long as the VM is fully utilized). While this increases the runtime on the cheaper VM, it proportionally reduces the execution time billed for the more expensive Lambda function. Provided that end-to-end latency and data privacy requirements are met, this strategic shift of computation from a high-cost to a low-cost resource can significantly reduce the total cost per inference. This demonstrates that resource selection and model partitioning must be carefully co-optimized to achieve a true balance among latency, privacy, and cost.

Despite progress in optimizing individual objectives, developing a unified framework to systematically balance inference latency, monetary cost, and privacy against Model

Inversion Attacks (MIA) remains a significant challenge. The current landscape reveals critical gaps in research. For instance, much of the existing work adopts a narrow definition of privacy, focusing on ensuring the physical locality of source data (e.g.,Multi-tier Multi-node Scheduling of LLM, etc [107, 114]) while overlooking the more subtle threat of information leakage from intermediate activations during inference (i.e., MIA and Prompt Inversion Attacks).

Furthermore, while the danger of Prompt Inversion Attacks (PIA) in Large Language Models is increasingly recognized, the practicality and efficiency of potential countermeasures are still largely unexplored. The remedies suggested by our privacy formulation, especially for LLMs, demand rigorous investigation before they can be considered viable. Drawing from these gaps in the literature and the challenges highlighted by our framework, the following section outlines several key open research issues.

6 Open Issues

Drawing from our multi-objective optimization framework, this section summarizes the inherent challenges in coupling latency, cost, and privacy, motivating future research directions.

6.1 Monetary Cost and Latency Optimization via Fine-Grained Resource Orchestration

Optimizing the monetary cost of hybrid edge-cloud ML systems introduces significant challenges in resource provisioning. Previous work, as discussed in Sec. 5.1, has primarily focused on load balancing at the **granularity of entire requests**. For instance, LIBRA [143] routes traffic to reserved VMs or serverless platforms based on traffic patterns by identifying a Cost Indifference Point, while Hybrid LLM [30] routes queries to different-sized models based on prompt difficulty.

We propose, however, that a more fine-grained approach, orchestrating resources at the level of **individual neural network partitions**, allows substantial cost savings, particularly for models featuring internal classifiers. When a DNN is partitioned, it forms a Directed Acyclic Graph (DAG) where each partition processes features for subsequent layers. The presence of early exits means that many requests may terminate at shallow partitions, leading to a significantly reduced and more variable workload for deeper ones.

Consequently, provisioning a single, powerful Virtual Machine (VM) for the entire inference path is inefficient. The resource remains idle for short-running requests but is billed as if fully utilized. A more cost-effective strategy would be to employ high-granularity, pay-per-use resources like FaaS for the deeper partitions. This allows the system to scale resources dynamically to match the fluctuating workload, ensuring that the monetary cost accurately reflects the actual computation performed.

6.2 Defending against Prompt Inversion Attacks under Latency and Cost Constraints

While our privacy formulation focuses on the general principles of Model Inversion Attacks (MIA), its application to Large Language Models (LLMs) is known as the **Prompt Inversion Attack (PIA)**. As LLM-based chatbots become popular, their prompts, often containing sensitive user data or proprietary business logic, have become valuable targets [113, 139]. However, traditional MIA remedies like regularization and perturbation are generally impractical for models of this scale, creating a substantial open attack surface for distributed LLM applications.

A difficulty in defending against PIA lies in the **unpredictable nature of privacy-preserving hyperparameter tuning**. As shown in our privacy formulation (Sec. 3.3), the final balance between model accuracy ($CE(\hat{y}, y)$) and privacy (invertibility of x_{pid-1}) for any given set of hyperparameters $\{\pi_{pid}^{pri}, \Delta, \lambda\}$ can only be evaluated after a full model training cycle converges.

This leads to two critical problems. First, the tuning process is **prohibitively time-consuming**, requiring numerous, resource-intensive training runs. Second, the tuning phase itself creates a **window of vulnerability**. In a distributed setting, the iterative exchange of gradients and intermediate activations during tuning exposes data *before* an effective privacy configuration has been found. While heuristics may offer some guidance (e.g., higher compression often degrades inversion accuracy), systematically and safely identifying optimal privacy hyperparameters for LLMs remains an unsolved research problem.

Some related work has attempted to mitigate these risks in traditional DNNs by using transfer learning [96]. The strategy is to pre-train a privacy-aware model on a public dataset and then fine-tune it on the private data, hoping to establish a secure baseline. However, this approach often fails when there is a significant domain mismatch between the public and private tasks (e.g., pre-training on CIFAR-10 for a CIFAR-100 task). In such cases, extensive fine-tuning on the private data is required to achieve acceptable accuracy, which reintroduces the original risks of high computational cost and data leakage during the prolonged tuning phase. Consequently, an efficient and secure methodology for tuning privacy-aware LLMs is critical.

7 Conclusion

This survey has contextualized state-of-the-art model offloading and adaptation methods as critical "control knobs" within a multi-objective optimization framework that balances latency, cost, and privacy. We traced the evolution from traditional full-model offloading to the sophisticated, partitioned neural network architectures required by demanding applications like LLM chatbots. The increasing prevalence of these applications highlights the significant potential for future edge-cloud collaborative Machine-Learning-as-a-Service (MLaaS) systems, particularly for organizations leveraging managed cloud infrastructure [2, 6, 185].

By analyzing common challenges such as transmission delays, processing overhead, and privacy vulnerabilities, this work provides a structured perspective on developing next-generation MLaaS platforms. The open issues identified, especially in minimizing monetary cost and strengthening guarantees against attacks like MIA and PIA, represent ground for future research. We believe this survey serves as a starting point to guide the future advancements in the field of collaborative edge-cloud intelligence.

References

[1] 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.

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

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

[4] Divyansh Agarwal, Alexander Fabbri, Ben Risher, Philippe Laban, Shafiq Joty, and Chien-Sheng Wu. Prompt leakage effect and mitigation strategies for multi-turn LLM applications. In Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1255–1275, Miami, Florida, US, November 2024. Association for Computational Linguistics. URL: https://aclanthology.org/2024.emnlp-industry.94/, https://doi.org/10.18653/v1/2024.emnlp-industry.94.

[5] 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.

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

[7] Amazon Web Services. Amazon EC2. https://aws.amazon.com/ec2/, 2024. Accessed: 2025-09-09.

[8] Amazon Web Services. Amazon ECS. https://aws.amazon.com/ecs/, 2024. Accessed: 2025-09-09.

[9] Amazon Web Services. Amazon Lambda. https://aws.amazon.com/lambda/, 2024. Accessed: 2025-09-09.

[10] Amazon Web Services. Amazon Lambda Edge. https://aws.amazon.com/lambda/edge/, 2024. Accessed: 2025-09-09.

[11] Amazon Web Services. Amazon Rekognition. https://docs.aws.amazon.com/rekognition/index.html, 2024. Accessed: 2025-09-09.

[12] Amazon Web Services. Amazon SageMaker. https://aws.amazon.com/sagemaker/, 2024. Accessed: 2025-09-09.

[13] Amazon Web Services. Amazon SageMaker Edge. https://aws.amazon.com/sagemaker/edge/, 2024. Accessed: 2025-09-09.

[14] Amazon Web Services. Amazon SageMaker NEO. https://aws.amazon.com/sagemaker/neo/, 2024. Accessed: 2025-09-09.

[15] Amazon Web Services. AWS IoT Greengrass. https://aws.amazon.com/greengrass/, 2024. Accessed: 2025-09-09.

[16] Amazon Web Services. AWS Local Zones. https://aws.amazon.com/about-aws/global-infrastructure/localzones/, 2024. Accessed: 2025-09-09.

[17] Amazon Web Services. AWS Wavelength. https://aws.amazon.com/wavelength/, 2024. Accessed: 2025-09-09.

[18] 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.

[19] Ons Aouedi, Thai-Hoc Vu, Alessio Sacco, Dinh C. Nguyen, Kandaraj Piamrat, Guido Marchetto, and Quoc-Viet Pham. A survey on intelligent internet

of things: Applications, security, privacy, and future directions. *IEEE Communications Surveys & Tutorials*, 27(2):1238–1292, 2025. https://doi.org/10.1109/COMST.2024.3430368.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] Yixin Chen, Shuai Zhang, Boran Han, and Bernie Wang. Visual Instruction Tuning with Chain of Region-of-Interest, 2025. URL: https://arxiv.org/abs/2505.06840, arXiv:2505.06840.

[26] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018. URL: https://arxiv.org/abs/1812.01718, arXiv:cs.CV/1812.01718.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] Alexey Dosovitskiy and Thomas Brox. Inverting Visual Representations with Convolutional Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4829–4837, 2016.

[34] 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.

[35] 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.

[36] 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.

[37] Yasuhiro Endo, Zheng Wang, J. Bradley Chen, and Margo Seltzer. Using latency to evaluate interactive system performance. In *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation*, OSDI '96, page 185–199, New York, NY, USA, 1996. Association for Computing Machinery. https://doi.org/10.1145/238721.238775.

[38] Ege Erdoğan, Alptekin Küpçü, and A. Ercüment Çiçek. UnSplit: Data-Oblivious Model Inversion, Model Stealing, and Label Inference Attacks Against Split Learning. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society (WPES '22)*, pages 115–124, Los Angeles, CA, USA, 2022. ACM. https://doi.org/10.1145/3559613.3563201.

[39] 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.

[40] 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.

[41] 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.

[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] 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.

[44] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 111–126, Santa Clara, CA, July 2024. USENIX Association. URL: https://www.usenix.org/conference/atc24/presentation/gao-bin-cost.

[45] Xinben Gao and Lan Zhang. PCAT: Functionality and data stealing from split learning by pseudo-client attack. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security '23)*, pages 5271–5288, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/gao.

[46] 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.

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

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

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] Alexey Guzey. How to Measure FLOP/s for Neural Networks Empirically? https://www.lesswrong.com/posts/jJApGWG95495pYM7C/how-to-measure-flop-s-for-neural-networks-empirically, September 2021. Accessed: 2025-09-09.

[56] 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.

[57] 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.

[58] 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.

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

[60] 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.

[61] 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.

[62] 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/.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating Pre-filling for Long-Context LLMs via Dynamic Sparse Attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL: https://openreview.net/forum?id=fPBACAbqSN.

[75] Huiqiang Jiang, Qianhui Wu, , Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL: https://aclanthology.org/2024.acl-long.91.

[76] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore, December 2023. Association for Computational Linguistics. URL: https://aclanthology.org/2023.emnlp-main.825, https://doi.org/10.18653/v1/2023.emnlp-main.825.

[77] 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.

[78] 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.

[79] 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.

[80] Dimitrios Kafetzis, Ramin Khalili, and Iordanis Koutsopoulos. Large language model partitioning for low-latency inference at the edge, 2025. URL: https://arxiv.org/abs/2505.02533, arXiv:2505.02533.

[81] 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.

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

[83] 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.

[84] 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.

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

[86] 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.

[87] 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.

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

[89] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. https://doi.org/10.1109/5.726791.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] 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.

[95] 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.

[96] 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.

[97] 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.

[98] Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV Cache-Centric Analysis of Long-Context Methods. In *The Thirteenth International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=gkUyYcY1W9.

[99] Yucheng Li, Huiqiang Jiang, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Amir H Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. MMIference: Accelerating Pre-filling for Long-Context VLMs via Modality-Aware Permutation Sparse Attention. In *Forty-second International Conference on Machine Learning*, 2025. URL: https://openreview.net/forum?id=me6PfbATWM.

[100] 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.

[101] 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. In *MLSys*, 2024.

[102] 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.

[103] 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.

[104] Barys Liskavets, Maxim Ushakov, Shuvendu Roy, Mark Klibanov, Ali Etemad, and Shane K. Luke. Prompt Compression with Context-aware Sentence Encoding for Fast and Improved LLM Inference. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025. https://doi.org/10.1609/aaai.v39i23.34639.

[105] 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.

[106] 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.

[107] 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.

[108] 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.

[109] 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.

[110] 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.

[111] 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.

[112] Daniel Lo, Taejoon Song, and G. Edward Suh. Prediction-guided performance-energy trade-off for interactive applications. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 508–520, 2015. https://doi.org/10.1145/2830772.2830776.

[113] Xinjian Luo, Ting Yu, and Xiaokui Xiao. Prompt inference attack on distributed large language model inference frameworks, 2025. URL: https://arxiv.org/abs/2503.09291, arXiv:2503.09291.

[114] Mulei Ma, Chenyu Gong, Liekang Zeng, and Yang Yang. Multi-tier multi-node scheduling of llm for collaborative ai computing. In *IEEE INFOCOM 2025 - IEEE Conference on Computer Communications*, pages 1–10, 2025. https://doi.org/10.1109/INFOCOM55648.2025.11044698.

[115] 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.

[116] 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.

[117] 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.

[118] 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.

[119] 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.

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

[121] 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.

[122] Microsoft. Guidance: A guidance language for controlling large language models. https://github.com/guidance-ai/guidance, 2022. Accessed: 2025-09-09.

[123] 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.

[124] 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.

[125] 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.

[126] 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.

[127] 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.

[128] Akrit Mudvari, Yuang Jiang, and Leandros Tassiulas. Splitllm: Collaborative inference of llms for model placement and throughput optimization, 2024. URL: https://arxiv.org/abs/2410.10759, arXiv:2410.10759.

[129] 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.

[130] 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/.

[131] 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.

[132] 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.

[133] OpenAI. Chatgpt. https://chat.openai.com, 2024. Accessed: 2025-09-09.

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

[135] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Ruhle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics ACL 2024*, pages 963–981, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. URL: https://aclanthology.org/2024.findings-acl.57.

[136] Dylan Patel and Afzal Ahmad. The Inference Cost Of Search Disruption – Large Language Model Cost Analysis. https://www.semianalysis.com/p/the-inference-cost-of-search-disruption, February 2023. Accessed: 2025-09-09.

[137] 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.

[138] 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.

[139] Wenjie Qu, Yuguang Zhou, Yongji Wu, Tingsong Xiao, Binhang Yuan, Yiming Li, and Jiaheng Zhang. Prompt inversion attack against collaborative inference of large language models. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1695–1712, 2025. https://doi.org/10.1109/SP61157.2025.00160.

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

[141] 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.

[142] 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.

[143] 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.

[144] Red Hat. Bring Insights and Data Closer to Customers with Edge Computing. White paper, Red Hat, February 2022. Accessed: 2025-09-09. URL: https://www.redhat.com/rhdc/managed-files/cl-bring-insight-data-customer-edge-computing-whitepaper-f30856pr-202202-en.pdf.

[145] Red Hat. 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, October 2023. Accessed: 2025-09-09.

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

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

[148] 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.

[149] 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.

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

[151] 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.

[152] David E. Rumelhart, James L. McClelland, and PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, 1986.

[153] 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.

[154] 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/.

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

[156] 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.

[157] 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.

[158] 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.

[159] 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.

[160] 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.

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

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

[163] 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.

[164] 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.

[165] Lin Song, Yukang Chen, Shuai Yang, Xiaohan Ding, Yixiao Ge, Ying-Cong Chen, and Ying Shan. Low-Rank Approximation for Sparse Attention in Multi-Modal LLMs. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13763–13773, 2024. https://doi.org/10.1109/CVPR52733.2024.01306.

[166] 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.

[167] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: dynamic scheduling for large language model serving. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, OSDI'24, USA, 2024. USENIX Association.

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

[169] Gemini Team et al. Gemini: A family of highly capable multimodal models, 2023. arXiv:2312.11805.

[170] 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.

[171] 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.

[172] 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.

[173] TensorFlow. Intro to Autoencoders. https://www.tensorflow.org/tutorials/generative/autoencoder, 2024. Accessed: 2025-09-09.

[174] Chunlin Tian, Xinpeng Qin, Kahou Tam, Li Li, Zijian Wang, Yuanzhe Zhao, Minglei Zhang, and Chengzhong Xu. Clone: customizing llms for efficient latency-aware inference at the edge. In *Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '25, USA, 2025. USENIX Association.

[175] 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.

[176] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *Proceedings on Privacy Enhancing Technologies*, 2021(2):323–343, 2021. https://doi.org/10.2478/popets-2021-0034.

[177] 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.

[178] 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.

[179] 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.

[180] 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.

[181] 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.

[182] Yiming Wang, Yu Lin, Xiaodong Zeng, and Guannan Zhang. Privatelora for efficient privacy preserving llm, 2023. URL: https://arxiv.org/abs/2311.14030, arXiv:2311.14030.

[183] 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.

[184] 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.

[185] 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.

[186] 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.

[187] Jinfeng Wen, Zhenpeng Chen, Jianshu Zhao, Federica Sarro, Haodi Ping, Ying Zhang, Shangguang Wang, and Xuanzhe Liu. Scope: Performance testing for serverless computing. *ACM Trans. Softw. Eng. Methodol.*, February 2025. Just Accepted. https://doi.org/10.1145/3717609.

[188] 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.

[189] 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.

[190] 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.

[191] 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.

[192] Xiph.Org Foundation. Xiph.org Video Test Media [derf's collection]. https://media.xiph.org/video/derf/, 2024. Accessed: 2025-09-09.

[193] Xiaoyang Xu, Mengda Yang, Wenzhe Yi, Ziang Li, Juan Wang, Hongxin Hu, Yong Zhuang, and Yaxin Liu. A Stealthy Wrongdoer: Feature-Oriented Reconstruction Attack against Split Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. Accessed: 2025-07-10. URL: https://arxiv.org/abs/2405.04115.

[194] 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.

[195] 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.

[196] 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.

[197] 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.

[198] 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.

[199] 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.

[200] 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.

[201] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, Vienna, Austria, July 2025. Association for Computational Linguistics. URL: https://aclanthology.org/2025.acl-long.1126/, https://doi.org/10.18653/v1/2025.acl-long.1126.

[202] 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.

[203] 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.

[204] Jintao Zhang, Haofeng Huang, Pengle Zhang, Jia Wei, Jun Zhu, and Jianfei Chen. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization. In *International Conference on Machine Learning (ICML)*, 2025.

[205] Jintao Zhang, Jia Wei, Pengle Zhang, Jun Zhu, and Jianfei Chen. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *International Conference on Learning Representations (ICLR)*, 2025.

[206] Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattn: Accurate sparse attention accelerating any model inference. In *International Conference on Machine Learning (ICML)*, 2025.

[207] 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.

[208] Shilong Zhang, Peize Sun, Shoufa Chen, Min Xiao, Wenqi Shao, Wenwei Zhang, Yu Liu, Kai Chen, and Ping Luo. GPT4RoI: Instruction Tuning Large Language Model on Region-of-Interest. In *Computer Vision – ECCV 2024 Workshops: Milan, Italy, September 29–October 4, 2024, Proceedings, Part VIII*, page 52–70, Berlin, Heidelberg, 2025. Springer-Verlag. https://doi.org/10.1007/978-3-031-91813-1_4.

[209] 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.

[210] 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.

[211] 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.

[212] 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.

[213] 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.

[214] 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.