DNN Partitioning for Cooperative Inference in Edge Intelligence: Modeling, Solutions, Toolchains

YUNTAO HAO, Dalian University of Technology School of Computer Science and Technology, Dalian, China
NAN DING, Dalian University of Technology School of Computer Science and Technology, Dalian, China
WEIGUO XIA, Dalian University of Technology School of Control Science and Engineering, Dalian, China
HONGWEI GE, Dalian University of Technology School of Computer Science and Technology, Dalian, China
LI XU, Beijing Bytedance Technology Co Ltd, Beijing, China

With rapid advancements in artificial intelligence and Internet of Things technologies, the deployment of deep neural network (DNN) models on the edge nodes and the end nodes has become an essential trend. However, the limited computational power, storage capacity, and resource constraints of these devices present significant challenges for deep learning inference. Traditional acceleration methods, such as model compression and hardware optimization, often struggle to balance real-time performance, accuracy, and cost-effectiveness. To address these challenges, collaborative inference through DNN partitioning has emerged as a promising solution. This article provides a comprehensive overview of architectural frameworks for DNN partitioning in collaborative inference. We establish a unified mathematical framework to describe various architectures, DNN models, and their associated optimization problems. In addition, we systematically classify and analyze existing partitioning strategies based on partition count and granularity. Furthermore, we summarize commonly used experimental setups and tools, offering practical insight into implementation. Finally, we discuss key challenges and open issues in DNN partitioning for collaborative inference, such as ensuring data security and privacy and efficiently partitioning large-scale models, providing valuable guidance for future research.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computing methodologies** → *Cooperation and coordination;*

Additional Key Words and Phrases: Collaborative inference, DNN partitioning, edge intelligence

ACM Reference Format:
Yuntao Hao, Nan Ding, Weiguo Xia, Hongwei Ge, and Li Xu. 2026. DNN Partitioning for Cooperative Inference in Edge Intelligence: Modeling, Solutions, Toolchains. *ACM Comput. Surv.* 58, 8, Article 204 (February 2026), 34 pages. https://doi.org/10.1145/3786145

Authors' Contact Information: Yuntao Hao (corresponding author), Dalian University of Technology School of Computer Science and Technology, Dalian, Liaoning, China; e-mail: haoyuntao@mail.dlut.edu.cn; Nan Ding (corresponding author), Dalian University of Technology School of Computer Science and Technology, Dalian, Liaoning, China; e-mail: dingnan@dlut.edu.cn; Weiguo Xia, Dalian University of Technology School of Control Science and Engineering, Dalian, Liaoning, China; e-mail: wgxiaseu@dlut.edu.cn; Hongwei Ge, Dalian University of Technology School of Computer Science and Technology, Dalian, Liaoning, China; e-mail: hwge@dlut.edu.cn; Li Xu, Beijing Bytedance Technology Co Ltd, Beijing, Beijing, China; e-mail: xuli.z@bytedance.com.

ACM 0360-0300/2026/02-ART204
https://doi.org/10.1145/3786145

1 Introduction

With rapid advances in AI [1] and IoT [2], **deep neural networks** (**DNNs**) [3] have been widely applied in fields such as image recognition [4], natural language processing [5], autonomous driving [6], and intelligent healthcare [7]. DNNs excel in classification, prediction, and decision-making due to their high accuracy. However, increasing task complexity demands larger models with more parameters and deeper architectures, leading to greater computational and storage requirements. For example, AlexNet [8] achieved improved image recognition accuracy with 8 layers and 60 million parameters; VGG-16 [9] (2014) employed 3×3 convolutions, extending to 16 layers and 140 million parameters; GPT-3 [10] (2020) has 175 billion parameters, requiring over 700 GB of memory and 3580 TFLOPs per inference.

Despite improvements in accuracy and efficiency, DNNs' computational and storage demands continue to grow. As DNN applications expand, inference bottlenecks intensify [11], especially in resource-limited edge environments with constrained computation, storage, and energy. In real-time applications like autonomous driving [12] and video analysis [13], the heavy computation often exceeds device capacities, causing latency and performance degradation. Therefore, optimizing inference efficiency while preserving accuracy is crucial. Various acceleration techniques have been proposed in academia and industry; these methods are summarized in [3] as follows:

(1) Model Optimization: Various techniques have been proposed to reduce DNN complexity while maintaining accuracy, such as quantization [14–16], pruning [17–19], knowledge distillation [20–22], and early-exit mechanisms [23, 24].

(2) Hardware Acceleration: Inference efficiency is further boosted through specialized hardware including GPUs [25], TPUs [26], FPGAs [27, 28], and ASICs [29, 30].

(3) DNN Partitioning: DNN partitioning reduces inference latency by leveraging the architecture's inter-layer connectivity and output size reduction. For instance, Tiny YOLOv2's input is 0.95 MB, while its max5 layer output is only 0.08 MB-a 93% decrease [31]. This size reduction minimizes data transmission and eases network bandwidth requirements. Partitioning distributes layers or sub-layers across nodes, balancing computation and communication through pipelined execution. As shown in Figure 1, Node 1 (resource-constrained, e.g., end or edge device) and Node 2 (more powerful, e.g., edge server or cloud) are linked via wired or wireless connections. The initial layers run on Node 1, with subsequent, heavier layers offloaded to Node 2. Intermediate feature maps are transmitted between nodes following the model's layerwise dependencies. By placing the partition where feature maps shrink significantly, transmission overhead is minimized without sacrificing accuracy, effectively balancing computation and communication costs.

While both model optimization and DNN partitioning target neural network processing, they differ fundamentally. Model optimization reduces model complexity by modifying the network structure through pruning, quantization, or knowledge distillation, often requiring extensive retraining and risking performance loss in complex tasks [32]. Such compressed models are typically platform- or dataset-specific, limiting generalizability. Hardware acceleration improves speed and energy efficiency via specialized hardware but faces deployment constraints in edge environments due to cost and power. Conversely, DNN partitioning retains the original model architecture, optimizing execution by distributing computation and communication across nodes (cloud, edge, end devices) based on their capabilities, making it well-suited for distributed, resource-limited scenarios that demand accuracy preservation [33].

1.1 Comparison and Our Contribution

Existing studies on DNN partitioning for collaborative inference explore diverse collaboration paradigms but often exhibit structural limitations. Early works focus on architectural classifications,

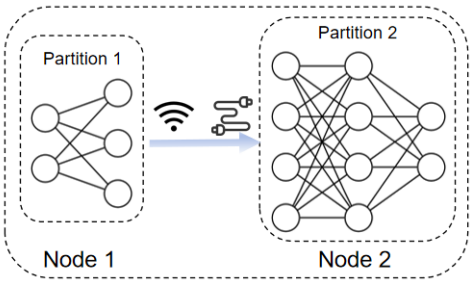


Fig. 1. DNN partitioning.

distinguishing between device-server and device-device collaboration [35, 38]. For instance, [38] surveys training and inference acceleration strategies in resource-constrained edge environments, including relevant metrics and supporting platforms. However, such taxonomies often lack the flexibility to capture real-world deployment heterogeneity. To address this, subsequent studies consider more comprehensive cloud-edge-end frameworks [37, 40, 41, 46]. Ref. [37] differentiates between cloud-edge-end and non-hierarchical partitioning approaches, while [46] emphasizes edge–cloud partitioning, where shallow layers are deployed on edge devices and deeper layers on the cloud. Other works [40, 41] propose hierarchical classifications based on the functional roles of cloud, edge, and end nodes. Nonetheless, these works tend to adopt coarse-grained structural views, overlooking variations within paradigms—e.g., one-to-multiple or multiple-to-one mappings—which result in distinct challenges such as resource scheduling or task offloading. Moreover, many categorize strategies by target objectives (e.g., latency, energy) rather than the underlying optimization formulations.

In contrast, our study adopts a problem-driven taxonomy that accounts for both architectural heterogeneity and model structure. We classify partitioning strategies based on their formulation characteristics—such as granularity selection, integration with model optimization, and dynamic adaptation—rather than surface-level metrics. This approach reveals deeper insights into how architectural choices and model properties jointly define the solution space. It also provides a more principled perspective on collaborative inference design.

Another line of research investigates collaborative DNN partitioning from the perspective of partitioning granularity. For example, [34] and [42] categorize partitioning strategies into horizontal and vertical partitioning. In this context, horizontal partitioning segments the model along the layer sequence—leveraging the sequential connectivity of DNN layers—whereas vertical partitioning divides the model at a finer granularity (e.g., sub-layer or neuron level), exploiting the inherent parallelism within layers. This granularity-based classification offers valuable insights into designing acceleration and optimization strategies for DNN inference in resource-constrained settings. Several other works explore collaborative inference under specific architectural contexts, emphasizing partitioning strategies and model optimization in IoT scenarios. Ref. [43] integrates collaborative architecture and partitioning granularity by classifying inference into three forms: device-server collaboration, horizontal partitioning between devices, and vertical partitioning between devices. This framework provides a theoretical basis for understanding collaborative mechanisms across different system layers.

Nevertheless, many of these studies tend to treat DNN partitioning as an isolated problem, overlooking its interdependence with other issues such as resource allocation [47]. In contrast, this article recognizes that DNN partitioning is not an isolated problem but is inherently coupled with other system-level and model-level challenges. For example, issues such as resource allocation,

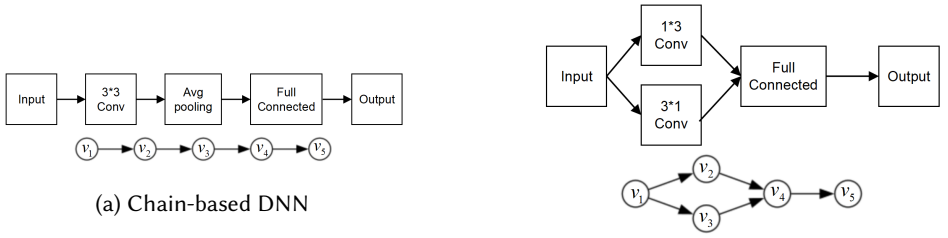
Table 1. Comparison with Related Works

Literature	Collaborative Inference Architecture	System Modeling & Optimization Problem	DNN Model Issues & Solutions	Transformer Partitioning Included	Collaborative Architecture Related Issues & Solutions	Experimental Tools & Datasets Summary
[34]	Cloud&Edge&End	✗	✓	✗	✗	✗
[35]	Device & Server	✗	✓	✗	✗	✗
[36]	–	✓	✗	✗	✗	✗
[37]	Cloud&Edge&End	✗	✗	✗	✗	✗
[38]	Device-server	✗	✗	✗	✗	✗
[39]	–	✓	✓	✗	✗	✗
[40]	Cloud&Edge&End	✓	✗	✗	✓	✗
[41]	Cloud&Edge&End	✓	✗	✗	✓	✗
[42]	Edge&End	✗	✓	✗	✗	✗
[43]	Device & Server	✗	✗	✗	✗	✗
[44]	–	✗	✓	✗	✗	✗
[45]	Cloud&Edge&End	✗	✓	✗	✗	✗
This paper	Cloud&Edge&End	✓	✓	✓	✓	✓

task offloading, and model optimization are tightly interwoven with partitioning decisions. This article explicitly models and analyzes these interdependencies—for instance, how resource availability constrains partitioning granularity, and how partitioning choices impact the distribution of computational load and communication cost. By incorporating these coupled factors into a unified optimization framework, the article provides a more holistic and realistic view of collaborative inference. Additionally, with the increasing prevalence of Transformer architectures, current surveys lack a systematic review of partitioning techniques tailored to Transformer-based models, which exhibit distinct structural and computational characteristics compared to traditional DNNs. To address this gap, this article reviews and analyzes recent Transformer-specific partitioning methods within the broader context of collaborative inference.

In this study, we compare our work with recent related research, as shown in Table 1. This survey aims to serve as a valuable reference for researchers to quickly grasp recent advancements and gain deeper insights into this evolving field. The primary contributions of this article are summarized as follows:

— **Unified Modeling Framework:** This article presents a systematic review and unified modeling of collaborative inference architectures, DNN structures, and their associated optimization problems. Beyond partitioning, it incorporates key system-level factors such as task offloading, resource allocation, mobility, and reliability, as well as model-level techniques including early exit, compression, and Transformer-specific strategies. The optimization objectives and constraints in prior work are abstracted into a unified formulation, providing a theoretical foundation for collaborative DNN inference design.
— **Solution Space-Oriented Classification:** This article proposes a novel classification framework based on the dimensionality of the solution space, encompassing six levels from single-layer partitioning to fine-grained tensor-level parallelism. It considers combinations with model optimization, resource allocation, and task offloading, enabling structured comparison of design tradeoffs and offering theoretical and practical guidance for diverse collaborative inference scenarios.
— **Practical Toolchain Overview:** This article surveys commonly used models, frameworks, datasets, hardware platforms, communication protocols, and evaluation tools in collaborative DNN partitioning research. It highlights their functionality, application scenarios, and limitations, offering a practical reference with official links to support reproducibility and real-world deployment.
— **Research Challenges and Open Issues:** This article outlines key challenges in collaborative DNN partitioning. These insights define a research agenda for advancing scalable, secure, and efficient collaborative inference systems.



(a) Chain-based DNN

(b) DAG-based DNN

Fig. 2. DNN classification.

1.2 The Organization of the Survey

Section 2 provides a detailed summary of scenario architecture, DNN models, and modeling optimization problem for DNN partitioning. It uses mathematical language to articulate each element, ensuring the rigor and unity of the theoretical framework. Section 3 systematically classifies and analyzes existing DNN partitioning methods from the perspectives of partition number and granularity. It also summarizes the application scenarios and characteristics of different partition strategies in collaborative inference. Section 4 presents the experimental setup and commonly used tools, providing insight into the practical implementation of DNN partitioning. Section 5 discusses the key challenges currently facing DNN partitioning collaborative inference and potential research directions. Finally, Section 6 concludes the article.

2 DNN Collaborative Inference: Model Structure, Scenario Architecture, and Optimization Issues

2.1 DNN Model Structure

To better understand the modeling of collaborative inference architectures and the associated optimization issues, it is essential to first examine how DNNs are structurally and computationally represented. As shown in Figure 2, a DNN consists of multiple layers, each containing a set of neurons responsible for receiving input, performing computations, and generating output [3].

Current studies have primarily focused on widely used DNNs such as **fully connected neural networks** (**FCNNs**), **convolutional neural networks** (**CNNs**), and Transformers. **Recurrent neural networks** (**RNNs**) and **generative adversarial networks** (**GANs**), these architectures have received comparatively less attention. To facilitate the design and deployment of partitioning algorithms, existing studies typically classify DNNs according to their topological structures rather than their functional categories. We denote a DNN inference task as G, and use $\mathcal{S}(G)$ to represent the storage requirement associated with G. The model structure is often fixed. However, different training datasets or input data can lead to varying inference tasks, which in turn result in different computational and storage demands.

As shown in Figure 2, based on the graph structure of the DNN, it can be categorized into two types: chain-based DNNs and DAG-based DNNs [48].

- Chain-based DNN: As shown in Figure 2(a), the network layers are arranged in a linear sequence, with the output of each layer serving as the input for the next layer. Well-known examples of chain-based DNNs include AlexNet [8] and VGG [9]. In some studies, Transformer models [49] are also treated as chain-based structures, due to their sequential layer-wise architecture and forward inference pattern [50–53].

- DAG-based DNN: As shown in Figure 2(b), the network structure can form a complex **directed acyclic graph** (**DAG**), allowing multiple paths to exist simultaneously, including potential

Table 2. Summary of References in One-to-One End-Edge Architecture

Reference	DNN Structure	Key Issues in DNN Partitioning
[59–62]	Chain	Early exit, Model compression
[63–68]	DAG	Model compression
[53]	Transformer	–

skip connections between different layers. Examples of DAG-based DNNs include ResNet [54], GoogleNet [55], and MobileNet [56].

Chain-based and DAG-based models rely on the explicit topological structure of the network to guide partitioning strategies. In contrast, an alternative class of methods focuses on the computational characteristics of individual layers, independent of the overall topology.

While both chain-based and DAG-based models rely on the explicit topological structure of the network to guide partitioning strategies, an alternative class of methods focuses on the computational characteristics of individual layers, independent of the overall topology.

- Topology-Agnostic DNN: Many layers in DNNs are composed of matrix-based operations, such as convolutions in CNNs [57] and multi-head self-attention and multilayer perceptrons in Transformers [58]. The inherent parallelism of matrix computation allows fine-grained inference parallelization. In such cases, DNN partitioning can be performed within individual layers rather than across layers.

We denote the partition set obtained from DNN partitioning as $\{G_1, G_2, \ldots, G_p\}$, where p denotes the number of partitions. This set can be derived either from layer-based partitioning or sub-layer-based partitioning. The computational cost of each partition is denoted as $\{Gcom_1, Gcom_2, \ldots, Gcom_p\}$. The operational storage requirements of each partition are denoted as $\{Stor_1, Stor_2, \ldots, Stor_p\}$. The initial data volume of the DNN is denoted as $Data_0$, while the output data volume for each partition is denoted as $\{Data_1, Data_2, \ldots, Data_p\}$. Specifically, early-exit branches are added to the DNN, and we denote the set of early-exit points as $\{Exit_1, Exit_2, \ldots, Exit_p\}$. These early-exit points allow the model to terminate inference at an intermediate layer once sufficient confidence is achieved.

2.2 Collaborative Inference Architecture in Edge Computing

(1) One-to-one End-edge Architecture: As shown in Figure 3, the end device both generates tasks and performs initial layer computations. It executes layers up to the partition point and uploads intermediate data to the edge server, which completes the remaining layers to produce the final result. Table 2 summarizes representative works, DNN structures, and key issues addressed in this architecture.

Different DNN architectures necessitate customized partitioning strategies. For chain-structured models, Neurosurgeon [59] identifies optimal partition points by balancing computational and communication costs across layers. For DAG-based models, DADS [63] formulates partitioning as a graph cut problem to address complex inter-layer dependencies. For **large language models (LLMs)**, [53] explores dynamic partitioning under variable wireless conditions to optimize edge-device collaboration.

Beyond standalone partitioning, several studies combine partitioning with model-level optimizations to enhance inference efficiency. Works such as [60, 61] integrate early exit mechanisms to adaptively terminate inference based on runtime conditions or confidence thresholds. This strategy reduces inference latency without compromising accuracy. Other efforts incorporate model compression: [67] applies compression prior to partitioning, while [68] performs partitioning first, followed by pruning on edge-deployed segments. These approaches highlight the interdependence

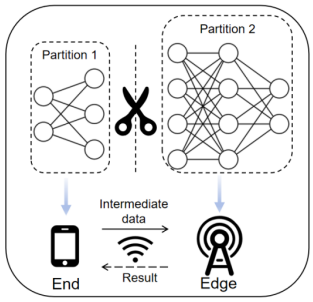


Fig. 3. One-to-one End-edge Architecture.

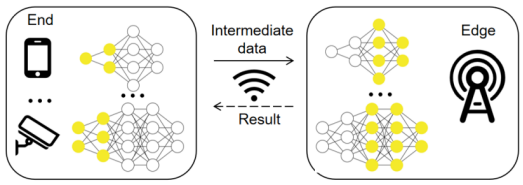


Fig. 4. Multiple-to-one end-edge architecture.

Table 3. Summary of References in Multiple-to-One End-Edge Architecture

Reference	Application Scenario	Key Issues Considered in DNN Partitioning
[33, 69–77]	IoT, Edge Computing	Resource allocation
[78–82]	Edge Computing	Task Offloading

between partitioning and optimization techniques in achieving efficient and adaptive DNN inference.

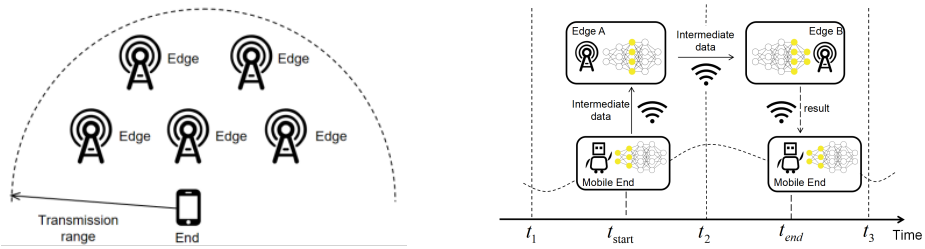
(2) Multiple-to-one End-edge Architecture: As shown in Figure 4, the multiple-to-one end-edge architecture is a widely adopted structure in collaborative DNN inference. In this architecture, multiple end devices offload their computation-intensive tasks to a shared edge server for processing. The edge server acts as a centralized coordinator that receives input data or partial model segments from various devices and completes the remaining stages of inference. Table 3 summarizes representative works, their corresponding application scenarios, and the key issues addressed in this architecture.

In the multiple-to-one end-edge architecture, two key challenges critically affect system performance: resource scheduling and DNN partition scheduling.

- **Resource Allocation:** Resource scheduling is critical as multiple end devices offload tasks to shared edge nodes, where overload(e.g., edge node utilization exceeding 90% [66]) leads to latency variability and degraded responsiveness. To address the interplay between workload distribution and limited system resources, studies have either jointly optimized DNN partitioning and resource allocation [33, 69–72, 76] or adopted decoupled solutions [73, 75, 81, 83].

- **Task offloading:** Under high concurrency, improper DNN partition scheduling can lead to severe resource contention, excessive queue delays, and suboptimal utilization of limited edge resources [77]. Improper scheduling in the case of multiple end devices offloading tasks to a single edge node can cause resource contention, increased queue delays, and inefficient resource utilization. The challenge lies in determining optimal partition points in the model based on task queue and communication conditions [77–80].

(3) One-to-multiple End-edge Architecture: The expansion of device communication range (Figure 5(a)) and mobility effects (Figure 5(b)) enable multiple edge servers to collaboratively serve a single end device. In this architecture, the DNN is partitioned into multiple segments, with the end device processing the initial segment locally and offloading the remainder to different edge nodes for distributed execution. Table 4 summarizes key works, application scenarios, and challenges in one-to-multiple end-edge architectures.



(a) Immobile (b) Mobility

Fig. 5. One-to-multiple end-edge architecture.

Table 4. Summary of References in One-to-Multiple End-Edge Architecture

Reference	Application Scenario	Key Issues Considered in DNN Partitioning
[84, 85]	Edge Computing	Task offloading
[85–88]	Mobile Edge Computing	Mobility-induced task offloading
[87, 89]	Vehicular Networks (V2I, V2V)	Reliability

Table 5. Summary of References in Multiple-to-Multiple End-Edge Architecture

Reference	Application Scenario	Key Issues Considered in DNN Partitioning
[90–92]	IoT, Edge Camera Network	Task Offloading, Resource Allocation
[47]	VEC	Mobility-induced Task Offloading Resource Allocation

Despite its advantages in leveraging multiple edge servers for distributed inference, the one-to-multiple architecture introduces several critical challenges that must be addressed to ensure efficient and reliable DNN partitioning and task execution.

- **Task Offloading:** The presence of multiple candidate servers complicates task offloading, requiring dynamic assignment of computation tasks alongside optimal DNN partitioning. Effective offloading must consider computational load, network conditions, and resource availability to avoid increased latency or node overload [85].

- **Mobility:** Mobility of user devices further exacerbates the difficulty of scheduling tasks. As network conditions—such as bandwidth, latency, and connectivity—fluctuate with device movement, task migration becomes increasingly frequent and unpredictable [85–88].

- **Reliability:** Reliability is critical in dynamic and intermittent wireless edge environments. Traditional collaborative partitioning and offloading methods, often rigid, struggle to adapt to real-time variations, causing reliability degradation [87, 88]. Mobility-induced disconnections and uneven resource distribution can interrupt inference. To improve robustness, [87, 89] propose overlapping DNN partitions, introducing redundancy by replicating critical model segments across nodes. This overlap balances computational efficiency with fault tolerance, enabling continued inference despite node failures or communication disruptions without full task re-execution.

(4) Multiple-to-multiple End-edge Architecture: In realistic edge computing deployments, many end devices within a limited geographic region often share access to multiple edge servers whose service coverage areas partially overlap. This results in a multiple-to-multiple architecture, where both end devices and edge servers are concurrently associated with multiple potential collaboration partners, as illustrated in Figure 6. Table 5 summarizes key works, scenarios, and issues in multiple-to-multiple end-edge architectures.

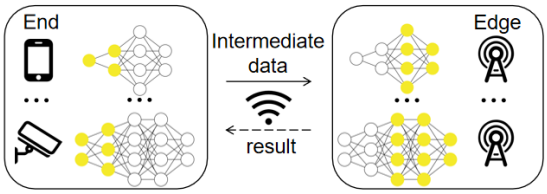


Fig. 6. Multiple-to-multiple architecture.

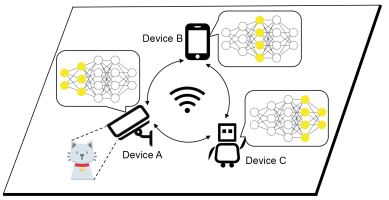


Fig. 7. Peer-to-peer architecture.

Table 6. Summary of References in Peer-to-Peer Architecture

Reference	Application Scenario	Key Issues Considered in DNN Partitioning
[32, 50–52, 57, 66, 93, 95, 99–104]	IoT, Edge Intelligence	Heterogeneity
[96]	UAV Swarm	Reliability
[97, 98]	Fog and Edge Computing	Task offloading
[94]	Vehicular Networks (V2V)	Mobility-induced task offloading

This architecture presents intertwined challenges in task offloading, resource allocation, and mobility [47, 74, 90–92]. To address these, many studies adopt decoupled optimization frameworks that separately solve partitioning, resource allocation, and offloading subproblems to reduce complexity [74, 90, 91]. Others, such as [92], decouple wireless **channel assignment** (**CA**) from joint DNN partitioning and resource allocation using graph-based clustering and two-stage optimization. More integrated approaches [47] jointly optimize all components in dynamic vehicular edge environments, capturing complex interdependencies to maximize long-term inference performance under mobility and resource constraints.

(5) Peer-to-Peer architecture: As illustrated in Figure 7, P2P architectures lack centralized control or hierarchy. In these systems, devices autonomously execute dynamically partitioned DNN segments, adapting to heterogeneity and network conditions [57, 93, 94]. Table 6 summarizes representative works, their corresponding application scenarios, and the key issues addressed in peer-to-peer architecture. Despite its flexibility and decentralized nature, the P2P collaborative inference architecture faces several critical challenges that distinguish it from more structured architectures such as one-to-multiple end-edge architecture.

- **Heterogeneity:** In P2P collaborative inference, device heterogeneity in computation, memory, energy, and hardware complicates task partitioning and load balancing, often causing inefficient resource use and suboptimal performance. Adaptive, fine-grained partitioning mechanisms are crucial. CoEdge [57] dynamically partitions workloads based on available computing and communication resources. Building on this, [95] generalizes from homogeneous to heterogeneous devices using an adaptive DAG-based partitioning algorithm for load balancing. Communication heterogeneity is addressed in [93] via fine-grained partitioning across devices with homogeneous servers but varying wireless channels. Recent studies [50–52] tackle partitioning of computation-intensive LLMs across distributed heterogeneous devices, formulating joint device selection and model partitioning problems.

- **Reliability:** The decentralized, dynamic nature of P2P systems increases the likelihood of device failures, link disruptions, and task interruptions compared to hierarchical architectures. Ensuring reliable inference under such uncertainties demands robust task replication, failure recovery, and fault-tolerant scheduling to maintain correctness despite partial failures. To address this, [96] enhances fault tolerance via multi-replica execution of DNN partitions across multiple UAVs operating in highly dynamic conditions.

- **Task Offloading:** In practical peer-to-peer (P2P) scenarios, multiple DNN inference tasks—ranging from chain-structured [97] to DAG-structured models [98]—are often generated

and processed concurrently across distributed devices. Under these conditions, suboptimal DNN partitioning and task offloading strategies can cause significant resource contention, excessive transmission and computation delays, and inefficient utilization of available bandwidth and computing resources.

- **Mobility:** Many devices in peer-to-peer systems—such as UAVs, vehicles, or mobile users [94]—are inherently mobile. Their movements lead to time-varying network topologies and intermittent connectivity, which in turn affect the continuity and timeliness of collaborative inference tasks. In such highly dynamic environments, rapid fluctuations in network topology and connectivity significantly increase the uncertainty of DNN task execution, making robust and adaptive scheduling essential.

Although this section describes various collaborative architectures in edge scenarios, it is important to note that architectures involving cloud nodes are also included. This encompasses cloud-end collaboration and cloud-edge collaboration, which can be categorized within the four types summarized in this section. We uniformly represent the computational nodes in these collaborative inference architectures as $\{N_0, N_1, \ldots, N_{\text{total}}\}$, where *total* is the total number of nodes. The computational power of each node is indicated as $\{com_0, com_1, \ldots, com_{\text{total}}\}$. The unit time computational energy consumption of each node is denoted as $\{\alpha_0, \alpha_1, \ldots, \alpha_{\text{total}}\}$. The transmission power of each node is indicated as $\{\beta_0, \beta_1, \ldots, \beta_{\text{total}}\}$. The unit operational cost of each node is denoted as $\{\gamma_0, \gamma_1, \ldots, \gamma_{\text{total}}\}$. The unit time transmission cost of each node is denoted as $\{\delta_0, \delta_1, \ldots, \delta_{\text{total}}\}$. The storage capacity of each node is indicated as $\{S_0, S_1, \ldots, S_{\text{total}}\}$. The bandwidth between any two nodes N_i, N_j is represented as B_{ij}, where $i, j = 0, 1, \ldots,$ total. If $i = j$, then $B_{ij} = \infty$. To account for system reliability, we model the node failure probability as $\{\phi_0, \phi_1, \ldots, \phi_{\text{total}}\}$, where $\phi_i \in [0, 1]$ represents the probability that node N_i fails during inference execution. Similarly, the failure probability of the transmission link between node N_i and node N_j is denoted by $\psi_{ij} \in [0, 1]$, which captures the likelihood of communication interruption on the corresponding channel.

2.3 Optimization Problems

In DNN partitioning for collaborative inference, the evaluation of **Quality of Service** (**QoS**) typically centers on five key metrics: latency, energy consumption, cost, accuracy, and reliability. To support QoS-aware DNN partitioning, accurate estimation of system parameters is essential. Existing literature generally categorizes the methods for obtaining these parameters into three main approaches:

- **Empirical Analysis:** Several studies obtain system parameters through empirical measurements. For instance, [105] notes that individual DNN layer execution times remain stable across runs on the same hardware. Building on this, works like [85, 86, 95, 105] measure layer-wise latency by executing models directly on target devices. For accuracy estimation, empirical evaluation is also prevalent: [71] assesses compressed DNN inference accuracy on standard public datasets.

- **Predictive Modeling:** This approach involves training predictive models using performance data collected under diverse network conditions, computational capacities, and system configurations. These trained models are then utilized during actual partitioning to estimate parameters with minimal overhead [59, 60]. In addition, some studies use models to predict accuracy. For instance, [61, 62, 106] leverage empirical modeling to estimate the expected accuracy of different early-exit branches at runtime.

- **Formula-based Estimation:** Many studies analytically compute system parameters through mathematical modeling. These formulations typically estimate the computational and communication costs of each DNN layer using layer-specific formulas.

Among the various parameter estimation techniques, analytical modeling stands out for its generality, transparency, and lightweight nature. Unlike empirical profiling or predictive modeling,

analytical formulas provide deterministic and reusable expressions derived from known system configurations and DNN structural properties. The following presents representative analytical formulations for key parameters, which form the basis of optimization frameworks and guide decision-making processes in collaborative DNN inference scenarios.

We use I_{ik} to indicate that the partition of the DNN inference task G_k is executed at the node N_i, and $I_{ik} = 0$ if it is not executed at that node N_i, where $i = 0, 1, \ldots,$ total and $k = 1, 2, \ldots, p$.Therefore, for any partition G_k, we can denote the node id of the computing node that executes the partition as $n_k = \sum_{i=0}^{\text{total}} i \cdot I_{ik}$. This results in a sequence of computational nodes $\{N_{n_1}, N_{n_2}, \ldots, N_{n_p}\}$ corresponding to the partition sequence $\{G_1, G_2, \ldots, G_p\}$.

(1) Latency Modeling: We use L to denote the total execution latency of a single DNN inference task, calculated as $L = L_c + L_t + L_{\text{que}}$.

L_c denotes computation latency of the node: $L_c = \sum_{k=1}^{p} \sum_{i=0}^{\text{total}} I_{ik} \cdot \frac{G_{\text{com}_k}}{com_i}$.

The parameter com_i represents the computational power of computational node N_i. In the literature, this parameter has been referred to by various terms: some studies use the term "computational power" [57], others refer to it as "server rate" [90], and several works adopt metrics such as floating-point operations per second (FLOPS) [65, 84, 97]. In this article, we adopt the unified term *computational power* to maintain consistency. In specific scenarios, different modeling strategies have been proposed to estimate com_i. For example, [32] models a multi-core processor as a single-core unit, with its computational power defined as the sum of its individual core speeds. Ref. [65] considers multi-threaded processing on edge servers and computes the overall computational power by multiplying the per-thread capability with the number of active threads. Furthermore, [69, 83] address the complexity of estimating the computational power of multi-core CPUs by introducing a compensation function. This function serves as a correction model to approximate the actual aggregated computing capability and is derived through nonlinear regression on empirically collected performance data. By doing so, it effectively captures the non-linear relationship between core count and total server-side computational power.

L_t denotes latency in data transmission: $L_t = \frac{Data_p}{B_{n_p n_1}} + \sum_{k=1}^{p-1} \frac{Data_k}{B_{n_k n_{k+1}}}$.

While bandwidth $B_{n_i n_j}$ is commonly used to estimate transmission latency, some studies further refine this estimation by incorporating channel conditions. Specifically, works such as [65, 67, 71, 72, 80, 83, 84, 92, 96] incorporate the Shannon–Hartley theorem to derive the channel capacity as a function of both bandwidth and **signal-to-noise ratio** (**SNR**), replacing the nominal bandwidth parameter with the theoretically achievable data rate. This approach enables a more accurate and information-theoretically grounded estimation of communication latency. Additionally, [80, 97] consider both uplink and downlink bandwidth conditions and adopt the minimum of the two as the bottleneck rate for latency calculation, further enhancing the precision of transmission modeling under asymmetric communication scenarios.

The last term of this equation denotes the transmission of the DNN inference results to the client device, and since the amount of data in the DNN inference results tends to be small, most studies often ignore the last term.

L_{que} denotes the queue delay: $L_{que} = \sum_{k=1}^{p} \sum_{i=0}^{\text{total}} I_{ik} \cdot L_{\text{que}_{ik}}$.

In computationally intensive tasks and high concurrency scenarios, some nodes may be resource constrained, causing inference tasks to queue up and wait, resulting in queuing delays [81].Queue waiting delay can be analyzed by queue theory models. Queue models such as M/M/1 and M/D/1 are commonly used to simulate the waiting time at various nodes during task processing. We use $L_{\text{que}_{ik}}$ to denote the queue waiting latency of DNN partition G_k on the computing node N_i.

(2) Energy Consumption Modeling: We use E to denote the total energy consumption of a single DNN inference task: $E = E_c + E_t$

E_c denotes computational energy consumption of the node: $E_c = \sum_{k=1}^{p} \sum_{i=0}^{\text{total}} I_{ik} \cdot \alpha_i \cdot \frac{G_{\text{com}_k}}{com_i}$.

In this formulation, the energy consumption is modeled as the product of computational latency and the node-specific energy consumption rate per unit time α_i. This ensures that the energy model maintains a direct and explicit dependency on latency, addressing the inherent correlation between delay and energy consumption. Several prior works adopt similar formulations. For instance, [57] models energy as the product of execution time and computational power. Other studies, such as [33, 72, 78], employ nonlinear models in which the dynamic power consumption is proportional to the cube of the CPU or GPU frequency, i.e., $\alpha_i \propto com_i^3$. Despite these variations in modeling details, latency remains a central factor in energy estimation.

E_t denotes data transmission energy consumption: $E_t = \beta_{n_k} \cdot \frac{Data_p}{B_{n_p n_1}} + \sum_{k=1}^{p-1} \beta_{n_k} \cdot \frac{Data_k}{B_{n_k n_{k+1}}}$.

In this formulation, β_{n_k} represents the transmission power, and energy consumption is calculated as the product of transmission power and transmission time. This structure ensures a direct latency-dependent energy model, consistent with practical communication energy estimation. Several studies adopt similar or extended approaches. For example, [57] estimates transmission energy as the product of transmission delay and device transmission power. Ref. [78] improves the transmission energy model by leveraging the Shannon–Hartley theorem to derive the effective data rate as a function of both bandwidth and SNR. The transmission energy is then calculated as the product of transmission power and the corresponding transmission time based on this refined rate.Despite superficial differences in notation or emphasis, these formulations are algebraically equivalent to the latency-based formulation used in this article. These diverse approaches all reinforce the notion that transmission energy is fundamentally a function of both data volume and the effective transmission rate, which itself depends on the communication environment and physical-layer parameters.

(3) Cost Modeling: We use C to denote the cost of a single DNN inference task: $C = C_c + C_t$.

C_c denotes computation cost of the node: $C_c = \sum_{k=1}^{p} \sum_{i=0}^{\text{total}} I_{ik} \cdot \gamma_i \cdot \frac{G_{\text{com}_k}}{com_i}$.

In this formulation, γ_i represents the unit operational cost associated with computational resources, and the computation cost is modeled as the product of execution latency and this unit cost. For example, [107–109] point out that according to the pay-as-you-go pricing model, the rental cost of a server depends on its price and inference latency. Furthermore, [75] proposes modeling γ_i as a function of the computational capacity com_i, thereby capturing the nonlinear relationship between computation cost and available resources more accurately.

C_t denotes data transmission cost: $C_t = \delta_{n_k} \cdot \frac{Data_p}{B_{n_p n_1}} + \sum_{k=1}^{p-1} \delta_{n_k} \cdot \frac{Data_k}{B_{n_k n_{k+1}}}$.

Here, δ_{n_k} represents the unit cost of utilizing the communication channel at node n_k. Similar to computational resources, the use of communication channels typically incurs costs, which may be associated with bandwidth usage, energy consumption, or service-level agreements [108]. Incorporating transmission cost into the system model allows for more comprehensive optimization that balances performance with economic considerations.

To illustrate the above models more concretely, consider a MobileNetV2 model with approximately 300 MFLOPs of total computation per inference. Suppose it is partitioned between a smartphone (Node 1) and an edge server (Node 2), with layers 1–5 (120 MFLOPs) executed on Node 1 and layers 6–10 (180 MFLOPs) offloaded to Node 2. The output of layer 5 produces a 2.5 MB feature map that needs to be transmitted to the server. The smartphone operates at 50 GFLOPS and the server at 500 GFLOPS, and the uplink bandwidth is 20 Mbps.

Latency model: Applying the latency model, the computation latency is 2.4 ms on Node 1 and 0.36 ms on Node 2. The transmission latency for the 2.5 MB intermediate data is $2.5 \times 8/20 = 1$ s, assuming a stable bandwidth. Furthermore, assuming a queueing delay at the edge server modeled as an M/M/1 queue with an arrival rate of 5 tasks/s and a service rate of 10 tasks/s, the expected queueing latency is 0.2 s. Therefore, the total end-to-end latency is approximately 1.203 s.

Energy model: If the smartphone's computational energy rate is 2 W and its transmission power is 1.5 W, the local computation energy is $2 \times 0.0024 = 4.8$ mJ, and the transmission energy is $1.5 \times 1 = 1.5$ J, yielding a total energy consumption of about 1.5048 J per inference.

Cost model: Assuming an electricity price of \$0.1/kWh, the computation cost is $\$\ 1.33 \times 10^{-10}$, and the transmission cost is $\$\ 4.17 \times 10^{-8}$, resulting in a total monetary cost of approximately $\$\ 4.18 \times 10^{-8}$ per inference.

(4) Accuracy Modeling: We use A to denote the accuracy of DNN inference: $A = F(Data_0, G, Exit_m)$.

Inference accuracy depends on input data size $Data_0$, DNN model structure G, and early exit branch points $Exit_m$ [110]. The relationship between these variables and inference accuracy is represented by a function F, which can be obtained through experimental measurements and regression analysis. In practice, the accuracy estimation often relies on two main approaches: empirical analysis and predictive modeling. Given the complexity of the underlying factors, we provide a rough functional approximation F to capture the general trend of accuracy variation with respect to these parameters.

(5) Reliability Modeling: We use R to denote the accuracy of DNN inference: $R = \prod_{k=1}^{p}(1 - \phi_{n_k}) \cdot \prod_{k=1}^{p-1}(1 - \psi_{n_k,n_{k+1}}) \cdot (1 - \psi_{n_p,n_1})$

It accounts for both computational node reliability and transmission link reliability [87, 89, 96]. Specifically, ϕ_{n_k} represents the failure probability of node N_{n_k}, and $\psi_{n_k,n_{k+1}}$ denotes the failure probability of the communication link between node N_{n_k} and node $N_{n_{k+1}}$. The final term ψ_{n_p,n_1} captures the reliability of the return link that transmits the final inference result back to the originating node N_{n_1}.

(6) Optimization Problems Modeling: Based on the above metrics, various types of optimization goals can be derived for DNN partitioning for collaborative inference.

$$\begin{aligned} \min \quad & \lambda_L L + \lambda_E E + \lambda_C C \\ \max \quad & A \\ \max \quad & R \\ \text{s.t.} \quad & \lambda_L + \lambda_E + \lambda_C = 1, \quad \lambda_L, \lambda_E, \lambda_C \in [0, 1] \end{aligned}$$

In the process of DNN partitioning collaborative inference, multiple constraints must be considered to ensure efficient and reliable execution of inference tasks. The common constraints are as follows:

$$\begin{aligned} &\text{C1: } p \le |G| \quad \vee \quad \textstyle\sum_{k=1}^{p} \dim(G_k) = \dim(G) \\ &\text{C2: } G_{com_k} \le com_{n_k} \\ &\text{C3: } L \le L_{\max} \\ &\text{C4: } Data_k \le B_{k,k+1} \cdot T_{\max} \\ &\text{C5: } A \ge A_{\min} \\ &\text{C6: } \mathcal{S}(G) + Stor_k \le S_{n_k} \end{aligned}$$

- Partition Feasibility Constraint (C1): For *layer-based partitioning*, the cuts must occur at valid layer boundaries, and the total number of partitions must not exceed the total number of layers in the model [59, 63]. For *sub-layer partitioning*, the model is divided at finer-grained levels such as

channels or tensor dimensions, where the partitioning must strictly preserve tensor compatibility across nodes [57].

- Computational Resource Constraint(C2): Each partition is assigned to a different computing node, and the computational load of the partition must not exceed the computational power of the node [47, 110].

- Latency Constraints(C3): DNN inference typically demands real-time performance. Consequently, the total latency of DNN inference should not exceed a predefined maximum delay L_{max}, which is dictated by the task deadline [80, 83].

- Communication Bandwidth Constraints(C4): The communication bandwidth between nodes is limited, and data transmission cannot occur without restrictions [84, 93]. Therefore, the intermediate data transmission time between nodes must remain within the permitted limit T_{max}.

- Accuracy Constraints (C5): Studies related to model compression ratio and early exit points often establishes stringent requirements on inference accuracy [62, 106].

- Storage Constraint (C6): Computing nodes typically need to store the complete DNN model [65, 80]along with the intermediate computational data generated during the DNN partitioning process. Therefore, it is crucial to ensure that each computing node has sufficient storage capacity to meet these requirements [57] .

Fundamentally, DNN partitioning refers to the decomposition of a DNN model at the layer or sub-layer level. When considering collaborative inference architecture, system-level characteristics-such as multi-node deployment, concurrent task execution, heterogeneity, reliability, and mobility-independently give rise to two major categories of optimization problems: task offloading and resource allocation. In contrast, the intrinsic architectural properties of DNN models—such as hierarchical structure, redundancy, and computational characteristics—naturally lead to model optimization and parallel processing problems. These two dimensions reflect different sources of complexity in collaborative inference and define distinct optimization perspectives.

From a mathematical perspective, standalone DNN partitioning problems are often formulated as **integer linear programming** (**ILP**) problems [58], where the solution space is defined by the hierarchical structure of the DNN layers or sub-layers. When extended to include decisions such as task offloading or model-level optimization (e.g., early exits), the solution space typically involves discrete variables (e.g., selection of offloading nodes, activation of exit branches) [62]. Although this increases computational complexity, the problem still falls within the ILP category. In contrast, resource allocation problems often involve continuous variables—such as the proportion of computational resources or communication bandwidth—which naturally extend the joint optimization of partitioning and resource allocation into a **mixed-integer nonlinear programming** (**MINLP**) problem [72]. These problems combine discrete decisions with nonlinear objectives or constraints, further increasing modeling and computational difficulty.

3 Method

Building on the summary of architectures and DNN modeling in Section 2, this section focuses on solution spaces and optimization methods for DNN partitioning (see Figure 8). It covers: one-dimensional spaces with single-layer partitioning; two-dimensional spaces integrating single-layer partitioning and model optimization (e.g., early exit, compression); joint multi-layer or sub-layer partitioning with task offloading; multi-dimensional spaces combining partitioning granularity, resource allocation, offloading, and model optimization; and fine-grained tensor parallelism-based partitioning. This systematic classification evaluates existing methods' applicability and effectiveness, providing a comprehensive overview of recent advances in DNN partitioning.

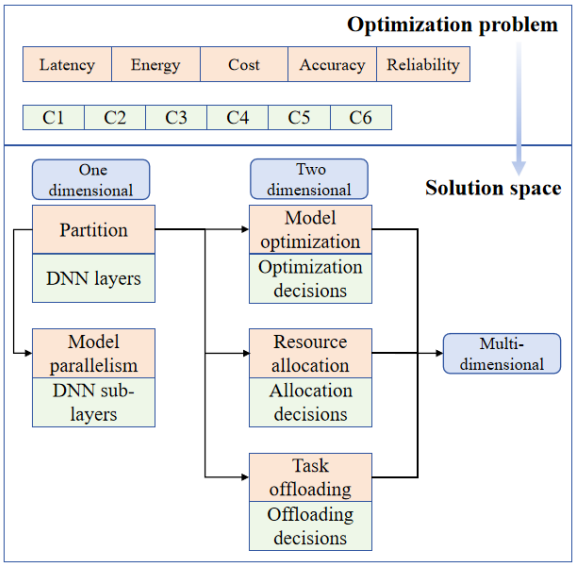


Fig. 8. Overall organization of method.

Table 7. Summary of Two-Partitioning Methods under Single-Layer Solution Space

Reference	Collaborative Architecture	Targeted Problem	Optimization Objective	Constraints	Method
[59, 72, 79]	One-to-One	Chain-structured DNN	Latency, Energy	C1, C3	Linear Search
[66, 77, 85, 86]		DAG-structured DNN	Latency	C1, C3	
[63, 65, 67, 111]	One-to-One	DAG-structured DNN	Latency	C1, C3	Graph-based Cut
[53]	One-to-One	Transformer	Latency,Accuracy	C1	DRL

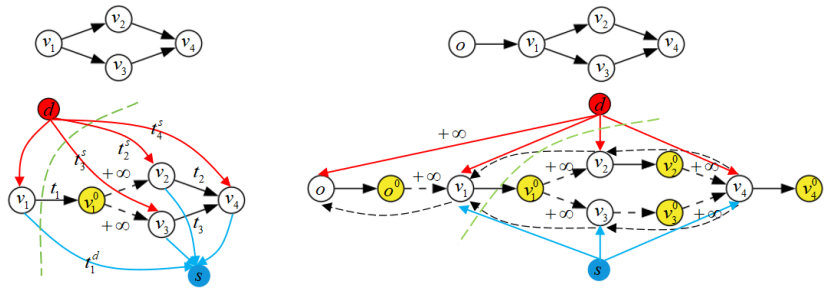
3.1 One-Dimensional Solution Space: Single-Layer Solution Space

When considering the partitioning problem independently, one of the most common challenges is how to divide the DNN model while accounting for its inherent topology. In such cases, the solution space is defined at the single-layer granularity. Based on how the partition point is determined, two-partitioning methods can be broadly classified into three categories: linear search methods, graph-based minimum cut methods and DRL methods. A summary of representative studies employing dual-partitioning strategies is presented in Table 7.

(1) Linear Search Method: The linear search method predominantly relies on predefined performance metrics, such as latency or energy consumption. This method involves analyzing the computational demands of the DNN and the costs associated with data transmission to evaluate the performance metrics of multiple candidate partition points. By comparing these metrics, the optimal partition point is selected to ensure effective performance while minimizing resource usage.

For chain-based DNNs, partitioning strategies typically aim to minimize latency by selecting optimal split points based on factors such as bandwidth, per-layer computational load, and node capacity [59, 66, 76, 79]. These methods often use linear search with iterative evaluation to converge toward optimal layer allocation. To jointly optimize latency and energy, [72] models partitioning as a non-cooperative game, computing FLOPs, latency, and energy for each strategy to identify a balanced tradeoff.

For DAG-based DNNs, researchers reduce complexity by approximating DAGs as sequential structures. Ref. [85] merges branches into a chain and applies shortest path algorithms to minimize



(a) Construction of graph G' (b) Improved construction of graph G'

Fig. 9. Graph-based minimum cut method.

latency, while [66] aggregates DAGs into modular blocks to reduce the complexity of min-cut partitioning. Ref. [77] further decomposes DAGs into chain-like branches and uses bisection search to iteratively balance computation and communication. These approaches enable efficient partitioning by simplifying DAG structures.

(2) Graph-based Cut Method: The linear search method is effective for small-scale or chain-structured DNNs due to its simplicity. However, its scalability is limited as model complexity increases. For DAG-based DNNs, as emphasized in DADS [112], partitioning becomes significantly more challenging due to complex inter-layer dependencies. To address this, graph-based minimum cut methods have been proposed, aiming to minimize computation and communication costs by strategically cutting the network graph.

As shown in Figure 9(a), DADS constructs an extended graph by adding virtual nodes. These nodes represent devices and servers, with auxiliary nodes used to capture communication delays. The partitioning problem is then transformed into a minimum cut problem. Cutting certain edges determines whether a layer is executed on the device or offloaded to the server. This approach effectively balances computation and communication by minimizing the total cut cost. DADS divides task loads into light and heavy categories. For light loads, the DSL algorithm uses graph min-cut to find optimal partitions, while heavy loads pose an NP-hard problem. The DSH algorithm addresses this by linearizing edge weights to obtain a local optimum efficiently. Ref. [67] highlights DADS's inefficiency on large models like InceptionV3 (313 layers), where partitioning exceeds one minute, failing real-time demands. To improve speed, [67] compresses the model before partitioning and employs the Dinic algorithm to accelerate min-cut computation. Ref. [65] identifies flaws in the DADS method's graph modeling of node computational capabilities, leading the minimum cut algorithm to offload entire DNNs to devices and ignore initial data upload time, resulting in poor partitioning. To fix this, [65] enhances graph construction by adding a virtual vertex o representing initial uploads and auxiliary vertices with infinite-weight edges to prevent unreasonable cuts, improving partitioning decisions (Figure 9(b)). Meanwhile, [111] extends graph-cut partitioning to jointly defend against GAN-based input reconstruction attacks and minimize total training time in collaborative learning. By splitting models between edge and cloud to protect activation privacy and assigning edge weights for forward/backward costs, the approach balances privacy and efficiency, demonstrating graph-based methods' versatility beyond inference acceleration.

(3) DRL Method: Cooperative inference typically involves multiple types of node computations and complex network transmission interactions. Factors such as network bandwidth change dynamically. These changes occur during actual operations. In this uncertain environment, the DRL

Table 8. Summary of Two-Partitioning Methods Integrating Partitioning with Model Optimization

Reference	Collaborative Architecture	Targeted Problem	Optimization Objective	Constraints	Method
[60]	One-to-One	Early Exit	Accuracy	C1, C3	Offline Configuration
[61]			Latency	C1, C5	Confidence Estimation
[62]			Latency	C1, C5	ILP Optimizer
[68]	One-to-One	Model Compression	Latency, Accuracy	C1, C5	Decoupled Optimization

method can effectively address dynamic partitioning issues by continuously learning and adjusting strategies through feedback. By perceiving the system's state in real time, the DRL method can adaptively select optimal partition points, thereby optimizing the overall performance of inference tasks.

The method in [53] comprehensively evaluates the impact of LLM partition points on inference performance under varying wireless channel conditions. It quantifies the inference performance of the LLM by considering the partition point , noise intensity , and the shape parameter , which directly affects the packet loss rate. A weighting factor is introduced to balance the tradeoff between inference performance and computational load. The determination of the optimal LLM partition point is formulated as a sequential decision-making process. **Proximal Policy Optimization (PPO)**, a reinforcement learning algorithm, is employed to adaptively select partition points.

3.2 Two-Dimensional Solution Space: Single-Layer and Model Optimization Solution Space

In addition to standalone partitioning strategies, several studies have explored the integration of DNN partitioning with model-level optimization techniques (e.g., early exit mechanisms and model compression) to further enhance inference efficiency in resource-constrained or dynamically changing environments. A comprehensive summary of these methods is presented in Table 8.

(1) Partitioning + Early Exit: [60] and [61] propose integrating DNN partitioning with early exit mechanisms to reduce inference latency and computational cost. Specifically, [60] constructs an offline configuration table that maps different bandwidth conditions to optimal partition points and early exit branches. At runtime, the system consults this table to select a partitioning and exit strategy based on current network conditions. Complementarily, [61] designs a confidence-based early exit mechanism: when the confidence score of an early exit branch exceeds a predefined threshold, the system accepts its output; otherwise, the inference continues by offloading the task to a more powerful server for further processing.

Going further, [62] introduces a fine-grained joint optimization framework that simultaneously considers partitioning, early exits, and intermediate feature compression. The optimization problem is formulated as an ILP model that selects the optimal combination of exit branch IDs, associated layer sets, and intermediate compression ratios, achieving a tradeoff between inference latency and accuracy under given resource constraints.

(2) Partitioning + Model Compression: [68] explores co-optimization approaches that combine model compression with DNN partitioning to alleviate computational loads on edge devices. Ref. [68] adopts a different sequence: partition points are selected based on layers with smaller output feature dimensions, and a sparsity-aware pruning method is subsequently applied to compress the submodel executed on edge nodes. This approach reduces edge-side computation while preserving inference accuracy, highlighting an effective and lightweight co-optimization strategy.

Table 9. Summary of Partitioning and Resource Allocation Methods under Two-Dimensional Solution Space

Reference	Collaborative Architecture	Targeted Problem	Optimization Objective	Constraints	Method
[73, 74]	One-to-Multiple	Resource Allocation	Energy	C1, C2	Decoupled Optimization
[75]			Latency, Energy	C1, C2	
[81]			Latency	C1, C2, C3	
[83]			Latency	C1, C2, C3	
[69]	One-to-Multiple	Resource Allocation	Latency	C1, C2	Iterative Alternating
[33]			Energy	C1, C2	DRL
[113]			Cost	C1, C2, C3	
[72, 76]			Latency, Energy	C1, C2	Game Theory

3.3 Two-Dimensional Solution Space: Single-Layer and Resource Allocation Solution Space

Approaches in this category can be broadly classified into decoupled optimization and joint optimization. A comprehensive summary of these methods is presented in Table 9.

(1) Decoupled Optimization: Rather than tackling DNN partitioning and resource allocation as a single integrated problem, a line of research addresses these tasks separately, thereby improving system modularity and reducing computational complexity. This decomposition allows for more focused and efficient solutions tailored to the unique characteristics of each subproblem.

For instance, [73] constructs an offline configuration table that maps different task types, model variants, parameter settings, and partition points to their corresponding computational and network resource requirements. Building upon this, constrained resource allocation in edge-based IoT systems is managed via an auction mechanism, effectively solving hierarchical decomposition challenges under communication and computation constraints. Similarly, [75] employs a minimum-cut algorithm to identify optimal DNN partition points first, before formulating the subsequent resource allocation as a game-theoretic problem. This sequential approach facilitates efficient and decoupled decision-making for both subproblems. A hybrid example can be found in [81], where reinforcement learning is leveraged to determine partition points, while a heuristic algorithm performs resource allocation, combining adaptive learning with practical scheduling strategies. Moreover, [83] proposes a binary decision tree structure (DP-tree) to systematically guide partitioning decisions. Following partitioning, communication and computational resources are asynchronously assigned to devices according to their traversal paths through the tree, thus explicitly decoupling partitioning logic from resource scheduling.

(2) Joint Optimization: Joint optimization treats DNN partitioning and resource allocation as a unified problem, enabling integrated decisions that capture their interdependencies for improved latency, energy, and resource efficiency in multi-user, multi-node collaborative inference. Unlike decoupled methods, this holistic approach better balances tradeoffs inherent in partition point selection and resource provisioning.

For instance, [69] analyze multi-user collaborative DNN partitioning and resource challenges, proposing an **Iterative Alternating Optimization** (**IAO**) algorithm that achieves optimality with polynomial complexity and robustness to estimation errors. To support real-time adaptation in dynamic environments, recent works apply **deep reinforcement learning** (**DRL**) [33, 113] to dynamically adjust partitioning and resources based on system feedback, enhancing efficiency in heterogeneous edge settings. Additionally, game-theoretic models [72, 76] frame joint optimization as strategic interactions among devices and edge nodes, enabling distributed, coordinated decisions that improve scalability and robustness under variable network and computational conditions.

Table 10. Summary of Multiple-Partitioning Methods: Integrating Partitioning with Offloading Decision

Reference	Collaborative Architecture	Targeted Problem	Optimization Objective	Constraints	Method
[84, 109]	One-to-Multiple	Task Offloading	Latency, Cost	C1	Decoupled Optimization
[86, 88]		Mobility	Latency	C1, C3	
[87, 89]		Reliability	Latency,Reliability	C1	
[93]	Peer-to-Peer	Task Offloading	Latency	C1, C4	
[85]	One-to-Multiple	Mobility	Latency	C1, C3	Algorithm-Based Method
[97, 98, 114]	Peer-to-Peer	Task Offloading	Latency	C1	
[115]	One-to-Multiple	Task Offloading	Latency	C1, C3	Heuristic Method
[99, 107]	Peer-to-Peer	Task Offloading	Latency, Energy	C1, C2, C6	
[78, 80, 108]	One-to-Multiple	Mobility	Latency, Energy, Cost	C1	Learning-Based Method
[94]	Peer-to-Peer		Latency, Energy	C1, C2, C3, C6	

3.4 Two-Dimensional Solution Space: Multiple-Layer/Sub-Layer and Offloading Decision Solution Space

For computationally intensive DNN models, multi-partitioning—dividing the network into several segments and distributing them across multiple nodes—offers greater flexibility and performance than traditional two-way partitioning [98]. Multi-partitioning is particularly relevant in scenarios involving: (1) heterogeneous nodes requiring partition allocation based on computational capabilities; (2) mobile end devices interacting with dynamic edge node availability; and (3) reliability demands, where redundant execution across nodes enhances fault tolerance. These scenarios require simultaneous optimization of multiple partition points and corresponding offloading decisions to minimize latency and avoid bottlenecks. Representative strategies are summarized in Table 10.

(1) Decoupled Optimization: These methods treat partitioning and offloading as independent subproblems to reduce complexity. In a cloud-edge-end architecture, [109] assumes ascending computational capabilities from end to cloud, selecting up to three partition points by evaluating latency and cost tradeoffs. If no optimal balance is found, latency is prioritized.

When the number of partitions is unconstrained, [93] proposes iterative multi-partitioning across multiple nodes. The offloading process is modeled as an **overlapping coalition formation** (**OCF**) game, solved via a genetic algorithm, enabling many-to-many user-server mappings. Similarly, [84] applies fine-grained sub-layer partitioning, using a decentralized matching game algorithm to optimize task assignment across fog nodes.

To reduce repartitioning overhead, [88] constructs a graph where vertices represent partition-offloading configurations, and edges connect neighboring decisions. A heuristic search leverages decision locality to accelerate convergence toward optimal solutions.

For reliability under unstable edge conditions, [87, 89] introduce replicated partitioning strategies. Ref. [89] balances latency and reliability via adaptive block merging and greedy consolidation. Ref. [87] develops a Markov chain-based reliability model, proposing approximate algorithms for partition-to-node mapping under uncertainty.

(2) Joint Optimization: In such methods, partitioning and offloading are treated as a unified problem, addressed through joint optimization.

- Algorithm-Based Methods: Layer-wise sequential decision-making is commonly adopted. For instance, [97] constructs a two-dimensional matrix encoding layer-node offloading decisions and applies DFS to find a minimal-latency execution path. Ref. [85] extends this by incorporating server-switching delays, enabling mobility-aware offloading under dynamic conditions.

To reduce complexity in DAG-structured DNNs, [98] performs topological sorting and applies greedy bipartite grouping, improving partition efficiency. Alternatively, [114] formulates the

Table 11. Representative Methods in Multi-Dimensional Solution Spaces

References	Architecture	Problem Scope	Optimization Objective	Constraints	Approach
[70, 71, 106]	Multiple-to-One	Joint management (1)	Latency	C1, C2, C5	DRL
[82]	Multiple-to-One	Joint management (2)	Latency, energy, accuracy	C1	DT-assisted DRL
[90, 91]	Multiple-to-Multiple	Joint management (3)	Latency, energy	C1, C2, C4, C6	Decoupled optimization
[92]			Latency	C1, C2	Partially decoupled joint optimization
[47]			Latency	C1, C2	

problem as a non-cooperative game among layers, where the proposed **Unit Competition of Layers (UCL)** algorithm converges to a Nash equilibrium through decentralized updates.

- Heuristic Methods: Heuristic approaches such as PSO-GA [115], enhanced GA [99], and **hybrid chaotic evolutionary algorithms (HCEA)** [107] are used to co-optimize latency, cost, and energy. These methods improve convergence and robustness through techniques like feasibility-aware genetic operations and chaotic population initialization.

- Learning-Based Methods: Learning-based joint optimization has emerged as a promising approach for DNN partitioning and offloading under dynamic and uncertain conditions.

To address mobility challenges, [108] introduces a DRL framework with a coordination-aware reward to balance latency, energy, and cost. In vehicular networks without RSUs, [94] proposes a two-stage DRL scheduler that adapts to rapid topology changes and handovers, ensuring stable inference. For environmental variability, [78] leverages local state information (e.g., channel quality, device capacity, transmission power) to guide partitioning decisions, while [80] incorporates latency, energy, and penalties into the reward to enable adaptive and efficient offloading. To enhance reliability, [96] presents a DQN-based method for partitioning and offloading across UAVs, meeting latency and reliability constraints. A multi-replica strategy is used to execute DNN segments on multiple UAVs concurrently, improving fault tolerance in dynamic environments.

3.5 Multi-Dimensional Solution Space: Integrating Partitioning Granularity, Resource Allocation, Task Offloading and Model Optimization

The intersection of these architecture-level and model-level issues creates a complex, highly multi-dimensional solution space encompassing partitioning granularity (at the layer or sub-layer level), offloading strategies, resource provisioning, and model optimization decisions. Based on the coupling patterns among these decision dimensions, existing methods can be broadly classified into three representative categories: joint management of partitioning, task offloading, and resource allocation, joint management of partitioning, model optimization, and task offloading, and joint management of partitioning, resource allocation, and task offloading. A comprehensive summary of these approaches is presented in Table 11.

(1) Joint management of partitioning, model optimization, and resource allocation: This integration is common in multiple-to-one end–edge collaborative architectures and is often addressed using DRL to enable adaptive, data-driven policy learning in dynamic, resource-constrained environments. For example, [70] formulates a multi-dimensional optimization. It jointly considers model partitioning, resource allocation, and early-exit selection for multi-exit DNN inference on heterogeneous mobile edge devices. To handle the combinatorial complexity, the MAMO framework uses bidirectional dynamic programming for optimal exit selection. A DRL component learns partitioning and resource allocation from offloading experience, enabling adaptive decisions across tasks. Similarly, [106] models joint partitioning, early exit selection, and resource distribution as a Markov Decision Process. It applies deep deterministic policy gradient to learn inference

policies in large, continuous state-action spaces, supporting real-time adaptation in multi-user settings.

In [71], a mixed-integer multi-dimensional optimization jointly selects DNN model versions, partitioning points, and computational and bandwidth resource allocation. This optimizes the balance between accuracy and latency. Multiple compressed DNN versions are trained in the cloud and deployed to end and edge devices. The DRL-based framework makes hierarchical decisions based on real-time network and resource states, selecting model versions, partition points, and resource assignments. This enables flexible inference strategies that optimize user experience while preserving accuracy.

(2) Joint management of partitioning, model optimization, and task offloading: [82] proposes a novel **digital twin** (**DT**)-assisted method that constructs a DT to evaluate potential offloading decisions for each DNN inference task. This provides enriched training data for machine learning-based decision algorithms. Additionally, another DT is built to estimate the inference status on the device, reducing frequent status queries and minimizing signaling overhead. For certain tasks, AIoT devices can opt for early exit from local DNN inference instead of offloading the entire task to the edge server. This approach dynamically determines whether and when to halt local inference and upload intermediate results to the edge server for completion, adapting to fluctuating workload conditions. By narrowing the offloading decision space and enabling multi-step dynamic decision-making, the method effectively balances inference accuracy and latency, particularly under heavy edge server loads where local early-exit prevents excessive latency caused by complete task offloading.

(3) Joint management of partitioning, task offloading, and resource allocation: This integration is common in multi-to-multi end–edge collaborative architectures, where managing interdependent decisions is complex. To address this challenge, several works adopt decoupled optimization. For example, [90] separates **Computing Resource Allocation** (**CRA**) with fixed partitioning from **DNN Partition Deployment** (**DPD**), solving CRA via a Markov approximation on a delay-aware cost and DPD with a polynomial-time near-optimal method. Similarly, [91] proposes a **Partitioning and Resource Allocation** (**PRA**) scheme for heterogeneous IoT devices, leveraging containerized DNN deployment and soft actor–critic DRL for dynamic resource allocation. Extending this, [92] decouples wireless CA using graph-based clustering, followed by two-stage optimization of partitioning and adaptive computing power allocation to minimize latency.

By contrast, [47] addresses tightly coupled vehicular edge computing by jointly optimizing offloading, partitioning, and resource allocation via Lyapunov-based reformulation and multi-agent diffusion DRL, better capturing the interplay of computation, communication, and mobility in dynamic heterogeneous environments.

3.6 One-Dimensional Solution Space: Tensor Parallelism Based on DNN Sub-Layers

As discussed in Section 2.1, the inherent parallelism of matrix computations enables fine-grained inference parallelization within DNNs. Sub-layer-based partitioning methods arise because dividing each DNN layer into smaller computational units facilitates more efficient parallel execution. We summarize selected works on sub-layer-based partitioning in Table 12.

(1) Parallel Partitioning for Convolutional Layers: MoDNN [117] and MeDNN [118] first introduced layer-wise parallelism for CNNs by partitioning convolutional feature maps into segments, with each node processing a portion and a main node aggregating outputs. However, this incurs high communication overhead. DeepThings [119] addresses this via **Fusion Tile Partitioning** (**FTP**) (Figure 10(a)), stacking convolution and pooling in tiled sublayers, enabling overlapping computations across nodes and significantly reducing data transfer and communication costs. AOFL [120] extends FTP by optimizing sublayer partitioning based on available compute resources,

(a) FTP (b) Only neighbor

Fig. 10. Partitioning of convolutional layers.

Table 12. Summary of Tensor Parallelism Based on DNN Sub-Layers Methods

Reference	Collaboration Architecture	Targeted Problem	Optimization Objective	Constraints	Approach
[116]	One-to-Multiple	Convolutional Layers Parallel	Latency	C1	VSM
[117, 118]	Peer-to-Peer			C1	Greedy Algorithm
[119]				C1	FTP
[120]				C1	AOFL
[57]			Energy	C1,C6	Only Neighbor
[112]	One-to-Multiple	Common DNN Layers Parallel	Latency	C1	Critical Path Method
[100, 121]	Peer-to-Peer			C1	Rearrangement of Neurons
[122]	Peer-to-Peer	Transformer Parallel	Latency	C1	Position-wise partitioning
[123]				C1	hybrid row- and column-wise
[58]				C1	WS & 1D & 2D

enhancing parallel efficiency. D3 [116] points out that DeepThings neglects padding in input feature maps, causing precision loss. To fix this, D3 introduces a **vertical separation module** (**VSM**) that preserves accuracy during parallel processing by properly handling padded feature maps and preventing error propagation.

The CoEdge [57] framework adopts a parallel workflow similar to that of DeepThings, where initial input is divided to allow multiple nodes to perform convolution and pooling operations for feature extraction. Finally, the extracted features are merged into a fully connected layer to produce the final output. However, CoEdge does not employ the layer fusion technique utilized by DeepThings. CoEdge highlights a potential issue in extreme scenarios where the convolution kernel is large, but the adjacent partition sizes are small. In such cases, the padding range may span three or more devices, resulting in additional communication overhead. To address this padding issue, CoEdge introduces a principle that requires the partition sizes allocated among neighboring devices to be no smaller than the padding size, unless no partitions are available. As shown in Figure 10(b), this principle ensures that padding data can always be sourced solely from adjacent devices whenever neighboring devices contain data, thereby reducing the communication overhead associated with data synchronization across devices.

(2) Parallel Partitioning for Common DNN Layers: Other works extend this concept to sub-layer partitioning across all DNN layers. For example, [100, 121] minimize interdependencies between parallel sub-models by progressively partitioning each layer-fixing the preceding layer's partitioning while optimizing the current one. As shown in Figure 11, this is achieved by rearranging neurons to reduce inter-layer dependencies and associated communication overhead (highlighted by red edges), effectively alleviating synchronization delays and enabling efficient distributed inference under resource constraints.

(3) Parallel Partitioning for Transformer: The sub-layer partitioning algorithms discussed above were originally developed for CNNs, mainly dividing tensors along the height dimension

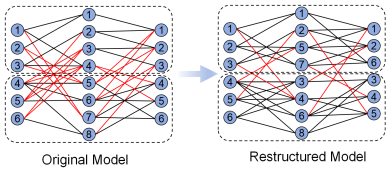


Fig. 11. Arbitrary layer partitioning.

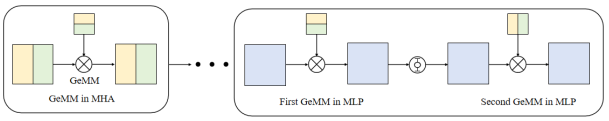
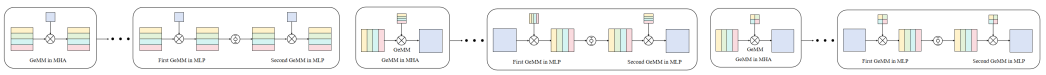


Fig. 12. Hybrid row and column in BP.



(a) WS (b) 1D (c) 2D

Fig. 13. Three parallel inference strategies in Hepti.

during convolution. However, Transformers differ fundamentally, consisting primarily of multi-head self-attention and feed-forward (MLP) blocks that rely heavily on matrix multiplication (MatMul/GeMM) operations. Consequently, CNN-specific partitioning methods do not directly apply to Transformers [58]. Early works such as [124, 125] parallelize Transformer models across multiple GPUs or TPUs by partitioning self-attention and MLP blocks. These approaches typically assume homogeneous computing nodes and rely on high-bandwidth interconnects for communication. However, such assumptions do not hold in edge environments, where heterogeneous devices and low-speed network links are common.

To reduce communication overhead from tensor parallelism and fully utilize multiple devices for lower inference latency, [123] proposes **block parallelism** (**BP**). As Figure 12 shows, weight matrices in **multi-head attention** (**MHA**) and the first MLP linear layer are partitioned row-wise, while the MHA output projection and second MLP layer are partitioned column-wise. This decouples layers by removing data dependencies among sub-blocks and defers communication until after several layers, with original layers serving as synchronization points. This design balances communication overhead and computational parallelism.

For edge deployments, [58] presents Hepti (Figure 13), where the main device estimates auxiliary devices' capabilities and bandwidth via prior inference timing. Hepti partitions GEMM operations in MHA and MLP, dynamically offloading tasks. It switches among three parallel strategies based on auxiliary memory:

— **Weight Stationary** (**WS**): partitions input feature matrices row-wise when memory suffices.
— 1D tiled WS: for fewer than 16 devices, partitions the first MHA weight matrix column-wise, the second row-wise; MLP input column-wise and weight row-wise.
— 2D tiled WS: for more devices, partitions MHA and MLP weight matrices by rows and columns; MLP input column-wise.

4 Experimental Setup and Commonly Used Tools

This section presents a comprehensive overview of the experimental environment and tools commonly employed in collaborative inference research.

4.1 Summary of DNNs, Associated Tools, and Datasets

This section presents an overview of the commonly used DNN models, associated frameworks, and datasets in collaborative inference.

Table 13. Popular DNN Frameworks and Common Datasets with Official URLs

Category	Tool or Dataset	URL
Framework	PyTorch	https://pytorch.org/
	TensorFlow	https://www.tensorflow.org/
	Caffe	https://caffe.berkeleyvision.org/
	BranchyNet	https://github.com/mit-han-lab/branchynet
Image Classification Datasets	CIFAR	https://www.cs.toronto.edu/~kriz/cifar.html
	Caltech-256	http://www.vision.caltech.edu/Image_Datasets/Caltech256/
	ImageNet	http://www.image-net.org/
	ILSVRC2012	http://www.image-net.org/challenges/LSVRC/2012/
	SeaShip	https://github.com/seaship-dataset/seaship
Video Datasets	BDD 100K	https://bdd-data.berkeley.edu/
	UCF-101	https://www.crcv.ucf.edu/data/UCF101.php
Text Classification Datasets	AG News	https://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html
	GLUE	https://gluebenchmark.com/
	WikiText-2	https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/

(1) DNN Models: In DNN partitioning experiments, models are generally categorized into three structural types:

— **Chain-based architectures:** Sequential, layer-wise models such as AlexNet, NiN, VGG, YOLO, MobileNet, and DarkNet.
— **DAG-based architectures:** Models with complex topologies that enable cross-layer information flow, e.g., ResNet, GoogLeNet, SqueezeNet, CharCNN.
— **Transformer-based architectures:** Attention-based models suitable for NLP and vision tasks, such as BERT [126], GPT-2 Medium, ViT [127], and LLaMA2 [128], which support tensor-parallel or sub-layer partitioning.

(2) Frameworks: Frameworks provide essential support for training, optimization, and experimental evaluation of DNN models. Common frameworks include PyTorch [129], TensorFlow [130], Caffe [131], BranchyNet [132], and Chainer [133].

(3) Datasets: Datasets offer standardized benchmarks for evaluating model performance, partitioning strategies, and optimization techniques. They can be categorized by task type:

— **Image classification:** CIFAR, Caltech-256, ImageNet, ILSVRC2012, SeaShip.
— **Video:** BDD 100K, UCF-101.
— **Text classification:** AG News, GLUE, WikiText-2.

Table 13 summarizes commonly used DNN frameworks and datasets in collaborative inference studies.

4.2 Summary of Computing Nodes and Communication Tools

This section presents an overview of computing nodes and communication tools commonly used in collaborative inference. Table 14 summarizes the key nodes and network communication tools along with their official URLs for reference and reproducibility.

Table 14. Computing Nodes and Network Communication Tools with Official URLs

Category	Node or Tool	URL
Server	Intel Xeon E5-2620 v4 / Intel i7-8700 / Intel i7-9700K / Intel i3-3240	—
	NVIDIA Titan V100 / RTX 2080 Ti / Quadro K620	—
Device	Raspberry Pi 3B / 3B+ / 4B / Raspberry Pi 4 Model B	—
	NVIDIA Jetson Nano / Xavier NX	—
Network Communication	WiFi	—
	Ethernet / LAN	—
	ZeroMQ	https://zeromq.org/
	gRPC	https://grpc.io/

Table 15. Resource Control Tools with Official URLs

Category	Tool or Dataset	URL
Bandwidth	WonderShaper	https://github.com/magnific0/wondershaper
	COMCAST	https://github.com/ANRGUSC/COMCAST
	Linux Traffic Control (tc)	https://man7.org/linux/man-pages/man8/tc.8.html
	Sleep Operation	https://man7.org/linux/man-pages/man1/sleep.1.html
	Belgium 4G/LTE Bandwidth Logs Dataset	https://github.com/ANRGUSC/COMCAST/tree/master/real-traces/belgium
Resource	Docker	https://www.docker.com/
Memory	stress-ng	https://manpages.ubuntu.com/manpages/focal/man1/stress-ng.1.html

(1) Computing Nodes:

- **Server Computing Nodes:** High-performance CPUs (Intel Xeon E5-2620 v4, Intel i7-8700/9700K, Intel i3-3240) and GPUs (NVIDIA Titan V100, RTX 2080 Ti, Quadro K620) with multithreading and large memory, suitable for large-scale DNN training and inference.
- **Device Computing Nodes:** Edge and embedded devices for lightweight inference, including Raspberry Pi Series (3B, 3B+, 4B, 4 Model B) and NVIDIA Jetson Series (Nano, Xavier NX).

(2) Network Communication Tools: Network communication tools are essential for enabling data exchange between computing nodes in collaborative DNN inference, supporting both lightweight edge tasks and high-bandwidth server operations. WiFi and Ethernet/LAN provide wireless and wired connectivity, respectively, catering to different bandwidth and latency requirements. ZeroMQ offers a message queue-based mechanism for efficient distributed communication, while gRPC is a high-performance RPC protocol designed for low-latency, high-throughput transmission between devices and MEC servers.

4.3 Summary of Resource Control Tools

This section provides an overview of resource control tools commonly used in collaborative inference. Table 15 summarizes the key resource control tools.

(1) Bandwidth Control Tools: These tools allow researchers to simulate network dynamics and evaluate their impact on DNN inference performance. Representative tools include WonderShaper,

Table 16. Parameter Analysis and Measurement Tools with Official URLs

Category	Tool	URL
Analysis Tool	TensorFlow Benchmarking Tool	https://www.tensorflow.org/guide/benchmarking
	PALEO	https://github.com/cucapra/paleo
	LINPACK	http://www.netlib.org/benchmark/linpackds/
Measurement Tool	thop	https://github.com/Lyken17/pytorch-OpCounter
	Torchstat	https://github.com/Swall0w/torchstat
	NetScope	https://netron.app/

COMCAST, Linux Traffic Control (tc), and Sleep Operation, while real-world datasets such as the Belgium 4G/LTE bandwidth logs provide dynamic traces for realistic experiments [60, 70, 97, 134].

(2) Resource Control Tools: Docker enables CPU resource allocation and parallel execution in containerized environments, supporting efficient management of computational resources in multi-core settings [69, 70, 134].

(3) Memory Control Tools: stress-ng simulates CPU and memory load, allowing evaluation of system behavior under constrained computational resources, commonly used in edge computing experiments [134].

4.4 Summary of Parameter Analysis and Measurement Tools

This section presents a summary of parameter analysis and measurement tools. Table 16 summarizes the principal parameter analysis and measurement tools introduced in this section.

(1) Analysis Tools: These tools evaluate inference performance and computational efficiency. Examples include TensorFlow Benchmarking Tool [57] for latency measurement, PALEO [99] for layer-wise performance and energy modeling, and LINPACK [62] for FLOPs benchmarking. They help quantify hardware capabilities but may have limitations in heterogeneous or resource-constrained edge environments.

(2) Measurement Tools: These tools assess model complexity, per-layer FLOPs, and memory usage. thop [97] and Torchstat [98] analyze PyTorch models, while NetScope [135] provides CNN visualization for architectural insights. They are informative for resource allocation, but may not fully capture dynamic runtime behavior or support large-scale and non-CNN models.

5 Challenges and Open Issues

5.1 Security and Privacy

DNN partitioning collaborative inference often involves heterogeneous nodes across cloud, edge, and end environments, some of which may be untrusted [136]. Resource-constrained edge and end devices are particularly vulnerable to physical attacks or malware [137]. Partitioning itself introduces additional risks, as intermediate activations transmitted between nodes may expose sensitive information. For example, model inversion attacks can reconstruct inputs by analyzing these intermediate activations [138], while adversarial perturbations injected by compromised nodes may mislead inference, such as misclassifying traffic signs [139]. Communication between partition points is also at risk from **man-in-the-middle (MITM)** attacks, which can intercept or alter transmitted data [140]. Although encryption offers protection, its overhead limits applicability in low-resource settings.

To mitigate these risks, lightweight security mechanisms–such as device authentication, **trusted execution environments** (**TEEs**), and tamper-proof storage–are needed. Blockchain can enhance trust by recording node behavior in a verifiable manner [141], while differential privacy protects input data from leakage [142]. However, existing privacy-preserving techniques like homomorphic encryption [143] and secure multiparty computation [144] often incur high overhead. Future work should focus on adaptive mechanisms that balance privacy and efficiency under resource constraints.

5.2 Robustness

Robustness is essential in DNN partitioning collaborative inference to ensure task completion amid uncertainties [145]. Failures of computation nodes or changes in system architecture, node status, or communication environments often require dynamic re-partitioning without interrupting inference [134]. As summarized previously, most existing works rely on multi-replica strategies, which improve fault tolerance by duplicating tasks but may introduce significant overhead and resource consumption.

The main challenge is designing fault-tolerant systems that can quickly migrate or reallocate tasks upon node failures, maintaining inference continuity [146]. Dynamic re-partitioning is critical for smooth task migration and scheduling coordination across nodes [147]. Automating rapid re-partitioning in response to node overload or failure will be vital to maintain uninterrupted inference services.

5.3 Partitioning of Large-Scale Models

Large-scale models, especially Transformers and GPT series, pose new challenges for collaborative inference due to their deep architectures and massive parameter sizes involving multi-head self-attention and feedforward layers. Although existing partitioning methods were not originally designed for such models, adaptations like layer-wise partitioning and pipeline parallelism have shown effectiveness [52, 58].

However, deploying these models on resource-constrained and heterogeneous devices such as mobile and edge platforms remains challenging. Issues include deep inter-layer dependencies, uneven computation across layers, and high memory usage. Moreover, communication overhead caused by transferring large intermediate data (e.g., attention maps) can offset the benefits of distributed execution. Future work should focus on resource-aware, hardware-adaptive partitioning strategies, leveraging runtime profiling, pipeline scheduling, and compression techniques to reduce communication costs while maintaining accuracy in heterogeneous collaborative environments.

6 Conclusion

This article presents a comprehensive survey of DNN partitioning for collaborative inference, addressing the challenges of deploying deep models on resource-constrained edge and end nodes. It unifies diverse collaborative architectures, DNN structures, and optimization objectives into a modeling framework and systematically compares partitioning strategies. Based on this analysis, several important directions for future research are highlighted. Ensuring data security and privacy at partition points, developing robust and adaptive partitioning strategies for dynamic and heterogeneous environments, and efficiently handling large-scale models are critical areas that require further exploration. In addition, lightweight and scalable solutions that balance latency, energy consumption, and monetary cost remain underexplored. We hope this work provides a theoretical foundation and practical reference to support further research and real-world deployment of collaborative DNN inference systems.

References

[1] Xuesong Zhai, Xiaoyan Chu, Ching Sing Chai, Morris Siu Yung Jong, Andreja Istenic, Michael Spector, Jia-Bao Liu, Jing Yuan, and Yan Li. 2021. A review of artificial intelligence (AI) in education from 2010 to 2020. *Complexity* 2021, 1 (2021), 8812542.

[2] Radouan Ait Radouan Ait Mouha et al. 2021. Internet of things (IoT). *Journal of Data Analysis and Information Processing* 9, 02 (2021), 77.

[3] Hanan Hussain, P. S. Tamizharasan, and C. S. Rahul. 2022. Design possibilities and challenges of DNN models: A review on the perspective of end devices. *Artificial Intelligence Review* (2022), 1–59.

[4] Yuxuan Liu, Hongwei Ge, Zhen Wang, Yaqing Hou, and Mingde Zhao. 2023. Clothes-changing person re-identification via universal framework with association and forgetting learning. *IEEE Transactions on Multimedia* (2023).

[5] Jan Sawicki, Maria Ganzha, and Marcin Paprzycki. 2023. The state of the art of natural language processing–A systematic automated review of NLP literature using NLP techniques. *Data Intelligence* 5, 3 (2023), 707–749.

[6] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. 2020. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access* 8 (2020), 58443–58469.

[7] Antonio Coronato, Muddasar Naeem, Giuseppe De Pietro, and Giovanni Paragliola. 2020. Reinforcement learning for intelligent healthcare applications: A survey. *Artificial Intelligence in Medicine* 109 (2020), 101964.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25 (2012), 1097–1105.

[9] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. Retrieved from https://arxiv.org/abs/1409.1556

[10] Min Zhang and Juntao Li. 2021. A commentary of GPT-3 in MIT technology review 2021. *Fundamental Research* 1, 6 (2021), 831–833.

[11] Ziheng Jiang, Tianqi Chen, and Mu Li. 2018. Efficient deep learning inference on edge devices. In *OSDI 2016 Conference Paper* (2018).

[12] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. 2020. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access* 8 (2020), 58443–58469.

[13] Himanshu Tyagi, Vivek Kumar, and Gaurav Kumar. 2022. A review paper on real-time video analysis in dense environment for surveillance system. In *Proceedings of the 2022 International Conference on Fourth Industrial Revolution Based Technology and Practices (ICFIRTP)*. IEEE, 171–183.

[14] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the International Conference on Machine Learning (ICML)* .

[15] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[16] Markus Nagel, Raoul Amjad, Mart van Baalen, Tijmen Blankevoort, and Max Welling. 2021. A white paper on neural network quantization. arXiv:2106.08295. Retrieved from https://arxiv.org/abs/2106.08295

[17] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural networks. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)* .

[18] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient convnets. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[19] Christos Louizos, Max Welling, and Diederik P. Kingma. 2018. Learning sparse neural networks through L_0 regularization. In *Proceedings of the International Conference on Learning Representations (ICLR)* .

[20] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. arXiv:1503.02531. Retrieved from https://arxiv.org/abs/1503.02531

[21] Ji Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* (2021).

[22] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. Fitnets: Hints for thin deep nets. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[23] Surin Teerapittayanon, Bradley McDanel, and H. T. Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *Proceedings of the ICASSP*.

[24] Simone Scardapane, Dian Wang, Yao Wang, and Aurelio Uncini. 2020. Should I leave this layer? A theoretical and practical perspective on early-exit strategies for deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[25] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable parallel programming with CUDA. *Queue* (2008).

[26] Normand P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*.

[27] Chen Zhang, Peng Li, Guangyu Sun, Yifan Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*.

[28] Stylianos I. Venieris and Christos-Savvas Bouganis. 2018. Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–39.

[29] Yu-Hsin Chen, Jeffrey S. Emer, and Vivienne Sze. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[30] Song Han, Xingyu Liu, Huizi Mao, Yu Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* .

[31] Ranjan Sapkota, Rizwan Qureshi, Marco Flores Calero, Muhammad Hussain, Chetan Badjugar, Upesh Nepal, Alwin Poulose, Peter Zeno, Uday Bhanu Prakash Vaddevolu, Hong Yan, et al. 2024. Yolov10 to its genesis: A decadal and comprehensive review of the you only look once series. arXiv:2406.19407. Retrieved from https://arxiv.org/abs/2406.19407

[32] Thomas Feltin, Léo Marchó, Juan-Antonio Cordero-Fuertes, Frank Brockners, and Thomas H. Clausen. 2023. DNN partitioning for inference throughput acceleration at the edge. *IEEE Access* 11 (2023), 52236–52249.

[33] Kaige Qu, Weihua Zhuang, Wen Wu, Mushu Li, Xuemin Shen, Xu Li, and Weisen Shi. 2023. Stochastic cumulative DNN inference with RL-aided adaptive IoT device-edge collaboration. *IEEE Internet of Things Journal* 10, 20 (2023), 18000–18015.

[34] Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. 2020. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 869–904.

[35] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE* 107, 8 (2019), 1738–1762.

[36] Ivan Rodriguez-Conde, Celso Campos, and Florentino Fdez-Riverola. 2023. Horizontally distributed inference of deep neural networks for AI-enabled IoT. *Sensors* 23, 4 (2023), 1911.

[37] Weixing Su, Linfeng Li, Fang Liu, Maowei He, and Xiaodan Liang. 2022. AI on the edge: A comprehensive review. *Artificial Intelligence Review* 55, 8 (2022), 6125–6183.

[38] Md Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I. Morshed. 2022. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE* 111, 1 (2022), 42–91.

[39] Emna Baccour, Naram Mhaisen, Alaa Awad Abdellatif, Aiman Erbad, Amr Mohamed, Mounir Hamdi, and Mohsen Guizani. 2022. Pervasive AI for IoT applications: A survey on resource-efficient distributed artificial intelligence. *IEEE Communications Surveys & Tutorials* 24, 4 (2022), 2366–2418.

[40] Wei-Qing Ren, Yu-Ben Qu, Chao Dong, Yu-Qian Jing, Hao Sun, Qi-Hui Wu, and Song Guo. 2023. A survey on collaborative DNN inference for edge intelligence. *Machine Intelligence Research* 20, 3 (2023), 370–395.

[41] Di Xu, Xiang He, Tonghua Su, and Zhongjie Wang. 2023. A survey on deep neural network partition over cloud, edge and end devices. arXiv:2304.10020. Retrieved from https://arxiv.org/abs/2304.10020

[42] Jiasi Chen and Xukan Ran. 2019. Deep learning with edge computing: A review. *Proceedings of the IEEE* 107, 8 (2019), 1655–1674.

[43] Sijing Duan, Dan Wang, Ju Ren, Feng Lyu, Ye Zhang, Huaqing Wu, and Xuemin Shen. 2022. Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. *IEEE Communications Surveys & Tutorials* 25, 1 (2022), 591–624.

[44] Federico Nicolás Peccia and Oliver Bringmann. 2024. Embedded distributed inference of deep neural networks: A systematic review. arXiv:2405.03360. Retrieved from https://arxiv.org/abs/2405.03360

[45] Yingchao Wang, Chen Yang, Shulin Lan, Liehuang Zhu, and Yan Zhang. 2024. End-edge-cloud collaborative computing for deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials* (2024).

[46] Xichen Zhang, Roozbeh Razavi-Far, Haruna Isah, Amir David, Griffin Higgins, and Michael Zhang. 2025. A survey on deep learning in edge-cloud collaboration: Model partitioning, privacy preservation, and prospects. *Knowledge-Based Systems* 310, Article 112965 (2025).

[47] Zhang Liu, Hongyang Du, Junzhe Lin, Zhibin Gao, Lianfen Huang, Seyyedali Hosseinalipour, and Dusit Niyato. 2024. DNN partitioning, task offloading, and resource allocation in dynamic vehicular networks: A Lyapunov-guided diffusion-based reinforcement learning approach. *IEEE Transactions on Mobile Computing* (2024).

[48] Shaohuai Shi, Qiang Wang, and Xiaowen Chu. 2018. Performance modeling and evaluation of distributed deep learning frameworks on GPUs. In *Proceedings of the 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 949–957.

[49] Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. 2022. Transformer for graphs: An overview from architecture perspective. arXiv:2202.08455. Retrieved from https://arxiv.org/abs/2202.08455

[50] Yang Hu, Connor Imes, Xuanang Zhao, Souvik Kundu, Peter A. Beerel, Stephen P. Crago, and John Paul Walters. 2022. Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices. In *Proceedings of the 2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE, 298–307.

[51] Ruilong Ma, Xiang Yang, Jingyu Wang, Qi Qi, Haifeng Sun, Jing Wang, Zirui Zhuang, and Jianxin Liao. 2024. HPipe: Large language model pipeline parallelism for long context on heterogeneous cost-effective devices. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*. 1–9.

[52] Mingjin Zhang, Xiaoming Shen, Jiannong Cao, Zeyang Cui, and Shan Jiang. 2024. Edgeshard: Efficient LLM inference via collaborative edge computing. *IEEE Internet of Things Journal* (2024).

[53] Yuxuan Chen, Rongpeng Li, Xiaoxue Yu, Zhifeng Zhao, and Honggang Zhang. 2024. Adaptive layer splitting for wireless LLM inference in edge computing: A model-based reinforcement learning approach. arXiv:2406.02616. Retrieved from https://arxiv.org/abs/2406.02616

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[55] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1–9.

[56] Andrew G. Howard. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861. Retrieved from https://arxiv.org/abs/1704.04861

[57] Liekang Zeng, Xu Chen, Zhi Zhou, Lei Yang, and Junshan Zhang. 2020. Coedge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Transactions on Networking* 29, 2 (2020), 595–608.

[58] Juhyeon Lee, Insung Bahk, Hoseung Kim, Sinjin Jeong, Suyeon Lee, and Donghyun Min. 2024. An autonomous parallelization of transformer model inference on heterogeneous edge devices. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 50–61.

[59] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.

[60] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2019. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications* 19, 1 (2019), 447–457.

[61] Roberto G. Pacheco, Rodrigo S. Couto, and Osvaldo Simeone. 2023. On the impact of deep neural network calibration on adaptive edge offloading for image classification. *Journal of Network and Computer Applications* 217 (2023), 103679.

[62] Sheng Hu, Chongwu Dong, and Wushao Wen. 2021. Enable pipeline processing of DNN co-inference tasks in the mobile-edge cloud. In *Proceedings of the 2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 186–192.

[63] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1423–1431.

[64] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing* 20, 2 (2019), 565–576.

[65] Jing Li, Weifa Liang, Yuchen Li, Zichuan Xu, Xiaohua Jia, and Song Guo. 2021. Throughput maximization of delay-aware DNN inference in edge computing by exploring DNN model partitioning and inference parallelism. *IEEE Transactions on Mobile Computing* 22, 5 (2021), 3017–3030.

[66] Hongzhou Liu, Wenli Zheng, Li Li, and Minyi Guo. 2022. Loadpart: Load-aware dynamic partition of deep neural networks for edge offloading. In *Proceedings of the 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 481–491.

[67] Run Yang, Yan Li, Hui He, and Weizhe Zhang. 2022. DNN real-time collaborative inference acceleration with mobile edge computing. In *Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 01–08.

[68] Mingran Li, Xuejun Zhang, Jiasheng Guo, and Feng Li. 2023. Cloud–Edge collaborative inference with network pruning. *Electronics* 12, 17 (2023), 3598.

[69] Xin Tang, Xu Chen, Liekang Zeng, Shuai Yu, and Lin Chen. 2020. Joint multiuser DNN partitioning and computational resource allocation for collaborative edge intelligence. *IEEE Internet of Things Journal* 8, 12 (2020), 9511–9522.

[70] Fang Dong, Huitian Wang, Dian Shen, Zhaowu Huang, Qiang He, Jinghui Zhang, Liangsheng Wen, and Tingting Zhang. 2022. Multi-exit DNN inference acceleration based on multi-dimensional optimization for edge intelligence. *IEEE Transactions on Mobile Computing* 22, 9 (2022), 5389–5405.

[71] Huamei Qi, Fang Ren, Leilei Wang, Ping Jiang, Shaohua Wan, and Xiaoheng Deng. 2024. Multi-compression scale DNN inference acceleration based on cloud-edge-end collaboration. *ACM Transactions on Embedded Computing Systems* 23, 1 (2024), 1–25.

[72] Xin Niu, Yajing Huang, Zhiwei Wang, Chen Yu, and Hai Jin. 2024. Game-based adaptive flops and partition point decision mechanism with latency and energy-efficient tradeoff for edge intelligence. *IEEE Transactions on Computers* (2024).

[73] Ya-Ting Yang and Hung-Yu Wei. 2021. Edge computing and networking resource management for decomposable deep learning: An auction-based approach. In *Proceedings of the 2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 108–113.

[74] Ya-Ting Yang and Hung-Yu Wei. 2022. A coalition formation approach for privacy and energy-aware split deep learning inference in edge camera network. *IEEE Transactions on Network and Service Management* 20, 3 (2022), 3673–3685.

[75] Xiang Yang, Dezhi Chen, Qi Qi, Jingyu Wang, Haifeng Sun, Jianxin Liao, and Song Guo. 2023. Adaptive DNN surgery for selfish inference acceleration with on-demand edge resource. arXiv:2306.12185. Retrieved from https://arxiv.org/abs/2306.12185

[76] Wangbing Cheng, MinFeng Zhang, Fang Dong, and Shucun Fu. 2023. Accelerate multi-view inference with end-edge collaborative computing. In *Proceedings of the 2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 1625–1631.

[77] Yubin Duan and Jie Wu. 2023. Optimizing job offloading schedule for collaborative DNN inference. *IEEE Transactions on Mobile Computing* 23, 4 (2023), 3436–3451.

[78] Feng Wang, Songfu Cai, and Vincent K. N. Lau. 2023. Sequential offloading for distributed DNN computation in multiuser MEC systems. *IEEE Internet of Things Journal* 10, 20 (2023), 18315–18329.

[79] Boyin Zhang, Yinggang Li, Shigeng Zhang, Yue Zhang, and Bing Zhu. 2022. An adaptive task migration scheduling approach for edge-cloud collaborative inference. *Wireless Communications and Mobile Computing* 2022, 1 (2022), 8804530.

[80] Jianbing Zhang, Shufang Ma, Zexiao Yan, and Jiwei Huang. 2023. Joint DNN partitioning and task offloading in mobile edge computing via deep reinforcement learning. *Journal of Cloud Computing* 12, 1 (2023), 116.

[81] Yi Su, Wenhao Fan, Li Gao, Lei Qiao, Yuan'an Liu, and Fan Wu. 2022. Joint DNN partition and resource allocation optimization for energy-constrained hierarchical edge-cloud systems. *IEEE Transactions on Vehicular Technology* 72, 3 (2022), 3930–3944.

[82] Shisheng Hu, Mushu Li, Jie Gao, Conghao Zhou, and Xuemin Sherman Shen. 2023. Adaptive device-edge collaboration on DNN inference in AIoT: A digital twin-assisted approach. *IEEE Internet of Things Journal* (2023).

[83] Huan Yang, Sheng Sun, Min Liu, Qiuping Zhang, and Yuwei Wang. 2023. MJOA-MU: End-to-edge collaborative computation for DNN inference based on model uploading. *Computer Networks* 231 (2023), 109801.

[84] Thaha Mohammed, Carlee Joe-Wong, Rohit Babbar, and Mario Di Francesco. 2020. Distributed inference acceleration with adaptive DNN partitioning and offloading. In *Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 854–863.

[85] Xianzhong Tian, Juan Zhu, Ting Xu, and Yanjun Li. 2021. Mobility-included DNN partition offloading from mobile devices to edge clouds. *Sensors* 21, 1 (2021), 229.

[86] Xiang He, Zhen Liu, Ze Kou, Ninghua Dong, Yiran Li, and Zeshuai Liu. 2023. HSMS-ADP: Adaptive DNNs partitioning for end-edge collaborative inference in high-speed mobile scenarios. In *Proceedings of the 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 1015–1022.

[87] Kai Liu, Chunhui Liu, Guozhi Yan, Victor C. S. Lee, and Jiannong Cao. 2023. Accelerating DNN inference with reliability guarantee in vehicular edge computing. *IEEE/ACM Transactions on Networking* 31, 6 (2023), 3238–3253.

[88] Bowen Pang, Sicong Liu, Hongli Wang, Bin Guo, Yuzhan Wang, Hao Wang, Zhenli Sheng, Zhongyi Wang, and Zhiwen Yu. 2023. AdaMEC: Towards a context-adaptive and dynamically combinable DNN deployment framework for mobile edge computing. *ACM Transactions on Sensor Networks* 20, 1 (2023), 1–28.

[89] Guozhi Yan, Chunhui Liu, and Kai Liu. 2023. ASPM: Reliability-oriented DNN inference partition and offloading in vehicular edge computing. In *Proceedings of the 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 3298–3303.

[90] Wenchen He, Shaoyong Guo, Song Guo, Xuesong Qiu, and Feng Qi. 2020. Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT. *IEEE Internet of Things Journal* 7, 10 (2020), 9241–9254.

[91] Taeyoung Kim, Hyungbin Park, Younghwan Jin, Seung-Seob Lee, and Sukyoung Lee. 2023. Partition placement and resource allocation for multiple DNN-based applications in heterogeneous IoT environments. *IEEE Internet of Things Journal* 10, 11 (2023), 9836–9848.

[92] Wenhao Fan, Li Gao, Yi Su, Fan Wu, and Yuan'an Liu. 2023. Joint DNN partition and resource allocation for task offloading in edge–cloud-assisted IoT environments. *IEEE Internet of Things Journal* 10, 12 (2023), 10146–10159.

[93] Tingting Yang, Hongjian He, and Lingzheng Kong. 2023. Multi-UAV maritime search and rescue with DNN inference acceleration. In *Proceedings of the 2023 International Conference on Ubiquitous Communication (Ucom)*. IEEE, 248–253.
[94] Yalan Wu, Jigang Wu, Mianyang Yao, Bosheng Liu, Long Chen, and Siew Kei Lam. 2023. Two-level scheduling algorithms for deep neural network inference in vehicular networks. *IEEE Transactions on Intelligent Transportation Systems* 24, 9 (2023), 9324–9343.
[95] Weiwei Miao, Zeng Zeng, Lei Wei, Shihao Li, Chengling Jiang, and Zhen Zhang. 2020. Adaptive DNN partition in edge computing environments. In *Proceedings of the 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 685–690.
[96] Mingyue Zhao, Xing Zhang, Zezhao Meng, and Xiangwang Hou. 2022. Reliable DNN partitioning for UAV swarm. In *Proceedings of the 2022 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 265–270.
[97] Guoliang Gao, Liantao Wu, Ziyu Shao, Yang Yang, and Zhouyang Lin. 2021. OCDST: Offloading chained DNNs for streaming tasks. In *Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.
[98] Liantao Wu, Guoliang Gao, Jing Yu, Fangtong Zhou, Yang Yang, and Tengfei Wang. 2023. PDD: Partitioning DAG-topology DNNs for streaming tasks. *IEEE Internet of Things Journal* (2023).
[99] Jun Na, Handuo Zhang, Jiaxin Lian, and Bin Zhang. 2022. Partitioning DNNs for optimizing distributed inference performance on cooperative edge devices: A genetic algorithm approach. *Applied Sciences* 12, 20 (2022), 10619.
[100] Afshin Abdi, Saeed Rashidi, Faramarz Fekri, and Tushar Krishna. 2023. Efficient distributed inference of deep neural networks via restructuring and pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 6640–6648.
[101] Maryam Ebrahimi, Alexandre da Silva Veith, Moshe Gabel, and Eyal de Lara. 2023. PORTEND: A joint performance model for partitioned early-exiting DNNs. In *Proceedings of the 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2475–2482.
[102] Suhwan Kim, Sehun Jung, and Hyang-Won Lee. 2023. Distributed computation of DNN via DRL with spatio-temporal state embedding. *IEEE Internet of Things Journal* (2023).
[103] Xiaotian Guo, Andy D. Pimentel, and Todor Stefanov. 2023. Automated exploration and implementation of distributed CNN inference at the edge. *IEEE Internet of Things Journal* 10, 7 (2023), 5843–5858.
[104] Penglin Dai, Biao Han, Ke Li, Xincao Xu, Huanlai Xing, and Kai Liu. 2024. Joint optimization of device placement and model partitioning for cooperative DNN inference in heterogeneous edge computing. *IEEE Transactions on Mobile Computing* (2024).
[105] Hongshan Li, Chenghao Hu, Jingyan Jiang, Zhi Wang, Yonggang Wen, and Wenwu Zhu. 2018. JALAD: Joint accuracy- and latency-aware deep structure decoupling for edge-cloud execution. In *Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 671–678.
[106] Xiaodong Xu, Kaiwen Yan, Shujun Han, Bizhu Wang, Xiaofeng Tao, and Ping Zhang. 2023. Learning-based edge-device collaborative DNN inference in IoVT networks. *IEEE Internet of Things Journal* (2023).
[107] Zengpeng Li, Huiqun Yu, Guisheng Fan, Jiayin Zhang, and Jin Xu. 2024. Energy-efficient offloading for DNN-based applications in edge-cloud computing: A hybrid chaotic evolutionary approach. *Journal of Parallel and Distributed Computing* 187 (2024), 104850.
[108] Min Xue, Huaming Wu, Guang Peng, and Katinka Wolter. 2021. DDPQN: An efficient DNN offloading strategy in local-edge-cloud collaborative environments. *IEEE Transactions on Services Computing* 15, 2 (2021), 640–655.
[109] Paridhika Kayal and Alberto Leon-Garcia. 2024. DNNSplit: Latency and cost-efficient split point identification for multi-tier DNN partitioning. *IEEE Access* (2024).
[110] Wenhao Fan, Zeyu Chen, Zhibo Hao, Yi Su, Fan Wu, Bihua Tang, and Yuan'an Liu. 2022. DNN deployment, task offloading, and resource allocation for joint task inference in IIoT. *IEEE Transactions on Industrial Informatics* 19, 2 (2022), 1634–1646.
[111] Cheng-Wei Ching, Tzu-Cheng Lin, Kung-Hao Chang, Chih-Chiung Yao, and Jian-Jhih Kuo. 2020. Model partition defense against GAN attacks on collaborative learning via mobile edge computing. In *Proceedings of the Globecom 2020-2020 IEEE Global Communications Conference*. IEEE, 1–6.
[112] Zhuofan Liao, Xiangyu Zhang, Shiming He, and Qiang Tang. 2023. PMP: A partition-match parallel mechanism for DNN inference acceleration in cloud–edge collaborative environments. *Journal of Network and Computer Applications* 218 (2023), 103720.
[113] Chongwu Dong, Sheng Hu, Xi Chen, and Wushao Wen. 2021. Joint optimization with DNN partitioning and resource allocation in mobile edge computing. *IEEE Transactions on Network and Service Management* 18, 4 (2021), 3973–3986.
[114] Jing Yu, Liantao Wu, Guoliang Gao, and Chenyu Gong. 2022. UCL: Unit competition of layers for streaming tasks in heterogeneous networks. In *Proceedings of the GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 5207–5212.
[115] Min Xue, Huaming Wu, Ruidong Li, Minxian Xu, and Pengfei Jiao. 2021. EosDNN: An efficient offloading scheme for DNN inference acceleration in local-edge-cloud collaborative environments. *IEEE Transactions on Green Communications and Networking* 6, 1 (2021), 248–264.

[116] Beibei Zhang, Tian Xiang, Hongxuan Zhang, Te Li, Shiqiang Zhu, and Jianjun Gu. 2021. Dynamic DNN decomposition for lossless synergistic inference. In *Proceedings of the 2021 IEEE 41st International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 13–20.

[117] Jiachen Mao, Xiang Chen, Kent W. Nixon, Christopher Krieger, and Yiran Chen. 2017. MoDNN: Local distributed mobile computing system for deep neural network. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 1396–1401.

[118] Jiachen Mao, Zhongda Yang, Wei Wen, Chunpeng Wu, Linghao Song, Kent W. Nixon, Xiang Chen, Hai Li, and Yiran Chen. 2017. MeDNN: A distributed mobile system with enhanced partition and deployment for large-scale DNNs. In *Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 751–756.

[119] Zhuoran Zhao, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. 2018. Deepthings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2348–2359.

[120] Li Zhou, Mohammad Hossein Samavatian, Anys Bacha, Saikat Majumdar, and Radu Teodorescu. 2019. Adaptive parallel execution of deep neural networks on heterogeneous edge devices. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 195–208.

[121] Chenghao Hu and Baochun Li. 2022. Distributed inference with deep learning models across heterogeneous edge devices. In *Proceedings of the IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 330–339.

[122] Chenghao Hu and Baochun Li. 2024. When the edge meets transformers: Distributed inference with transformer models. In *Proceedings of the 2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 82–92.

[123] Jiangsu Du, Yuanxin Wei, Shengyuan Ye, Jiazhi Jiang, Xu Chen, Dan Huang, and Yutong Lu. 2024. Co-designing transformer architectures for distributed inference with low communication. *IEEE Transactions on Parallel and Distributed Systems* (2024).

[124] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems* 5 (2023), 606–624.

[125] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv:1909.08053. Retrieved from https://arxiv.org/abs/1909.08053

[126] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

[127] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv:2010.11929. Retrieved from https://arxiv.org/abs/2010.11929

[128] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. arXiv:2307.09288. Retrieved from https://arxiv.org/abs/2307.09288

[129] A. Paszke. 2019. Pytorch: An imperative style, high-performance deep learning library. arXiv:1912.01703. Retrieved from https://arxiv.org/abs/1912.01703

[130] Abadi Martín, Agarwal Ashish, Barham Paul, Brevdo Eugene, Chen Zhifeng, Citro Craig, S. Corrado Greg, Davis Andy, Dean Jeffrey, Devin Matthieu, et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. *Software Available from Tensorflow. org* 7 (2015).

[131] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*. 675–678.

[132] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *Proceedings of the 2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2464–2469.

[133] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: A next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, Vol. 5. 1–6.

[134] Ayesha Abdul Majeed, Peter Kilpatrick, Ivor Spence, and Blesson Varghese. 2021. NEUKONFIG: Reducing edge service downtime when repartitioning DNNs. In *Proceedings of the 2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 118–125.

[135] Penglin Dai, Biao Han, Ke Li, Xincao Xu, Huanlai Xing, and Kai Liu. 2024. Joint optimization of device placement and model partitioning for cooperative DNN inference in heterogeneous edge computing. *IEEE Transactions on Mobile Computing* (2024).

[136] Latif U. Khan, Ibrar Yaqoob, Nguyen H. Tran, S. M. Ahsan Kazmi, Tri Nguyen Dang, and Choong Seon Hong. 2020. Edge-computing-enabled smart cities: A comprehensive survey. *IEEE Internet of Things Journal* 7, 10 (2020), 10200–10232.

[137] Abdullatif Ghallab, Mohammed H. Saif, and Abdulqader Mohsen. 2021. Data integrity and security in distributed cloud computing–A review. In *Proceedings of the International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications: ICMISC 2020*. Springer, 767–784.

[138] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1322–1333.

[139] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xia, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1625–1634.

[140] Lei Yu, Ximeng Liu, Yixian Yang, and Jianfeng Ma. 2019. MitM attacks on federated learning: Challenges and research opportunities. *IEEE Network* 33, 6 (2019), 162–167.

[141] Massimo Di Pierro. 2017. What is the blockchain? *Computing in Science & Engineering* 19, 5 (2017), 92–95.

[142] Ying Zhao and Jinjun Chen. 2022. A survey on differential privacy for unstructured data content. *ACM Computing Surveys (CSUR)* 54, 10s (2022), 1–28.

[143] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. 2018. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–35.

[144] Hanrui Zhong, Yingpeng Sang, Yongchun Zhang, and Zhicheng Xi. 2020. Secure multi-party computation on blockchain: An overview. In *Proceedings of the 10th International Symposium on Parallel Architectures, Algorithms and Programming, PAAP 2019, Revised Selected Papers 10*. Springer, 452–460.

[145] Xiaotian Guo, Quan Jiang, Andy D. Pimentel, and Todor Stefanov. 2024. RobustDiCE: Robust and distributed CNN inference at the edge. In *Proceedings of the 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 26–31.

[146] Ayesha Abdul Majeed, Peter Kilpatrick, Ivor Spence, and Blesson Varghese. 2022. CONTINUER: Maintaining distributed DNN services during edge failures. In *Proceedings of the 2022 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE, 143–152.

[147] Francis McNamee, Schahram Dustdar, Peter Kilpatrick, Weisong Shi, Ivor Spence, and Blesson Varghese. 2021. The case for adaptive deep neural networks in edge computing. In *Proceedings of the 2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 43–52.

Received 27 March 2025; revised 20 July 2025; accepted 7 November 2025