Accelerating End-Cloud Collaborative Inference via Near Bubble-free Pipeline Optimization

Luyao Gao[1,2], Jianchun Liu[1,2], Hongli Xu[1,2], Sun Xu[1,2], Qianpiao Ma[3], Liusheng Huang[1,2]
[1]School of Computer Science and Technology, University of Science and Technology of China, China
[2]Suzhou Institute for Advanced Research, University of Science and Technology of China, China
[3] School of Computer Science and Engineering, Nanjing University of Science and Technology, China

IEEE INFOCOM 2025 - IEEE Conference on Computer Communications | 979-8-3315-4305-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/INFOCOM55648.2025.11044632

***Abstract*—End-cloud collaboration offers a promising strategy to enhance the Quality of Service (QoS) in DNN inference by offloading portions of the inference workload from end devices to cloud servers. Despite the potential, the complex model architectures and dynamic network conditions will introduce numerous bubbles (*i.e.*, idle waiting time) in pipeline execution, resulting in inefficient resource utilization and degraded QoS. To address these challenges, we introduce a novel framework named COACH, designed for near bubble-free pipeline collaborative inference, thereby achieving low inference latency and high system throughput. Initially, COACH employs an *offline* component that utilizes an efficient recursive divide-and-conquer algorithm to optimize both model partitioning and transmission quantization, aiming to minimize the occurrence of pipeline bubbles. Subsequently, the *online* component in COACH employs an adaptive quantization adjustment and a context-aware caching strategy to further stabilize pipeline execution. Specifically, COACH analyzes the correlation between intermediate data and label semantic centers in the cache, along with its influence on the quantization adjustment, thereby effectively accommodating network fluctuations. Our experiments demonstrate the efficacy of COACH in reducing inference latency and enhancing system throughput. Notably, while maintaining comparable accuracy, COACH achieves up to 1.7× faster inference and 2.1× higher system throughput than baselines.**

***Index Terms*—Collaborative Inference, Bubble-free, Model Partition, Quantization Adjustment.**

I. Introduction

Traditionally, the DNN-based Artificial Intelligence (AI) applications (*e.g.*, model inference tasks) are usually offloaded to powerful cloud servers for advanced processing, facilitating faster and more accurate DNN inferences [1]. However, transmitting data to a server raises privacy concerns, particularly with sensitive information. Additionally, many resource-limited end devices struggle with the demands of complex DNN tasks, such as real-time high-definition video processing, which underscores a capability gap for less powerful devices [2]. In order to address the disparity in computing resources, collaborative inference involving both end devices and cloud servers has emerged, enabling DNNs to be segmented for distributed processing [3]–[5]. The process initiates with the end device executing the initial DNN segment and sending the intermediate data to the server, which then completes the processing and returns the final result [6].

Current approaches of collaborative inference always aim at the following two goals. (1) **Low inference latency.** Quick decision-making is crucial in time-sensitive applications such as autonomous vehicles, which necessitate the swift processing of sensor data within 20ms [7]. To facilitate this, DNN models are typically partitioned into two segments for parallel processing, with task latency comprising three parts: end device computation latency, intermediate data transmission latency, and server computation latency [6]. To achieve low latency, it is essential to explore efficient DNN partitioning and compression strategies that take into account the diverse capabilities of end devices and servers [8], [9]. (2) **High system throughput.** In practice, end devices often need to process continuous inference tasks, necessitating efficient scheduling strategies to enhance the throughput of inference systems [3]. Meanwhile, due to complex models and high workloads, the latency of the cloud computation stage cannot be ignored and should also be considered in scheduling [1]. To this end, pipeline parallelism is a crucial technique, allowing for the overlap of computation and transmission stages [10]. However, inference efficiency will be compromised by numerous pipeline bubbles (*i.e.*, idle waiting time), caused by unbalanced stage execution times [11]. Thus, reducing bubbles during pipeline execution is crucial for enhancing the inference performance.

However, two major challenges related to model partition and transmission extremely hinder the reduction of bubbles. (1) **Complex model architecture.** The evolution of DNN designs from linear to more sophisticated structures, such as Directed Acyclic Graphs (DAGs) exemplified by GoogleNet [12] and ResNet [13]. The multiple dependencies in DAG models create exponential search spaces for optimal partitioning and compression strategies, complicating pipeline parallelism [4]. Moreover, the intricate connections within these models often introduce more bubbles in the pipeline, further dragging the inference process [14], [15]. (2) **Dynamic network conditions.** The effectiveness of collaborative inference is substantially influenced by network conditions, which are inherently variable in real-world applications [16]. Although carefully crafted scheduling strategies can minimize latency in the pipeline execution, numerous bubbles still arise due to dynamic networks, resulting in high transmission latency and reduced system throughput [17].

Existing collaborative inference approaches primarily aim at reducing task latency and improving system throughput from the following two aspects. First, some works [4], [5], [18]

focused on optimizing the DNN partition strategy for complex models, yet they inadequately tackled the issues related to pipeline parallelism across varied computing resources of both end devices and server. Second, other works [11], [19], [20] concentrated on parallel processing to manage the multiple dependencies and complex interactions in DAG models, but they generally failed to account for the emergence of bubbles in pipeline execution of continuous tasks. This oversight frequently leads to the accumulation of numerous bubbles in the pipeline. Additionally, these approaches were typically designed for static conditions and experienced substantial performance degradation under dynamic networks [14].

To address the aforementioned challenges, we present COACH, a novel collaborative inference framework designed for near bubble-free pipeline execution, including an *offline* component and an *online* component. COACH is engineered to achieve low latency and high throughput under dynamic network conditions by minimizing bubbles in the pipeline. It incorporates an *offline* component that layer-wisely determines the joint model partitioning and transmission quantization strategy, and carefully manages layer parallel execution. Subsequently, the *online* component facilitates real-time assessment of the status of task features, enabling an adaptive quantization adjustment and a context-aware caching strategy to stabilize the pipeline execution in dynamic networks.

Nevertheless, to achieve efficient collaborative inference performance, two principal challenges still need to be addressed in COACH. First, we observe that different layers within DNN models require varied levels of quantization precision to satisfy specific accuracy criteria [4], as shown in Section II-B. This observation highlights the complex and tight interaction between partitioning and quantization strategies, further complicated by layer dependencies and pipeline parallelism. Consequently, it is critical to *determine the optimal partitioning and quantization strategy* to achieve bubble-free pipeline execution. Second, if pipeline scheduling fails to adapt to network fluctuations, the increase in transmission latency may lead to substantial pipeline bubbles, underscoring the need for quantization adjustment to stabilize transmission. Thus, another challenge involves how to *implement adaptive quantization adjustment in dynamic networks* to enhance collaborative inference performance. The key contributions of this paper are as follows:

- We present COACH, a novel collaborative inference framework designed for near bubble-free pipeline execution. To the best of our knowledge, COACH is the first framework that integrates the *offline* and *online* components to optimize the collaborative inference scheduling by sufficiently minimizing pipeline bubbles.
- The *offline* component utilizes a novel recursive divide-and-conquer algorithm to jointly determine optimal model partitioning and transmission quantization strategies, thereby achieving balanced and efficient computation and transmission phases.
- The *online* component implements an adaptive quantization adjustment and a context-aware caching strategy to accommodate network fluctuations. It adjusts the quantization precision according to the correlation between intermediate data and the semantic centers in the cache.
- Extensive experimental results demonstrate that COACH significantly surpasses existing collaborative inference approaches. Notably, while maintaining comparable accuracy, COACH achieves up to 2.1× faster inference and 2.5× higher throughput compared to baselines.

II. Background and Motivation

A. Related Works

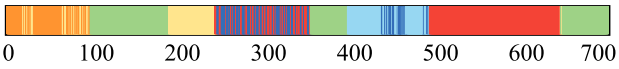
The collaborative inference paradigm, which leverages strategic workload offloading, aims to expedite the DNN inference process across heterogeneous end devices and servers [3]–[6]. Optimization techniques generally involve model partitioning, transmission compression, and dynamic scheduling.

A significant advancement in *model partitioning* is provided by Neurosurgeon [6], which reduces latency through tailored partitioning strategies for chain topology models. IONN [21] adopts the shortest path algorithm to enhance simultaneous inference processing, representing a novel methodological shift. OfpCNN [22] advances model partitioning with a fine-grained approach that optimally utilizes heterogeneous resources on devices and servers. Nonetheless, model partitioning introduces an increased transmission overhead, potentially complicating the partitioning search space [23], [24]. Recent advancements in DNN *transmission compression* techniques, such as quantization, have been directed towards tackling the challenge of high transmission overhead, thereby reducing transmission requirements [4], [5], [8], [18]. Approaches like CNNPC [5] and auto-split [18] employ quantization to enhance the potential for collaborative inference execution, though they may encounter challenges in adaptation across diverse end-cloud environments.

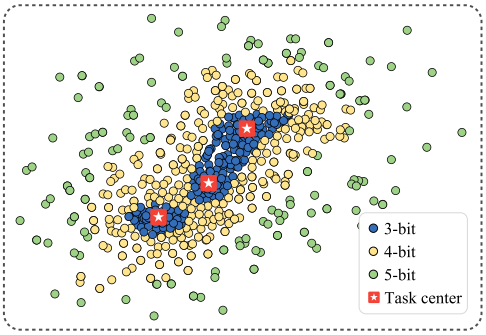
Additionally, focusing exclusively on optimizing a single task is inadequate for managing continuous task scenarios. It is essential to concurrently schedule multiple inference tasks for efficient execution, particularly when employing common practice pipeline scheduling [17], [25]. The *scheduling complexity* inherent in collaborative execution across devices and servers, particularly within DAG topology models, has emerged as a critical area of focus [3], [20], [26], [27]. Hu *et al.* [3] and Li *et al.* [19] explore the complexities of optimal offloading scheduling for DAG models in continuous task scenarios, yet their approaches primarily focus on static environments and tend to increase the occurrence of bubbles in dynamic networks. Duan *et al.* [20] and Zeng *et al.* [28] investigate DNN partitioning and pipeline parallelism strategy, focusing on throughput enhancement for DAG models. However, their approaches primarily optimize pipeline scheduling for parallel execution, while neglecting the cloud computation stage and the latency tolerance of individual inference tasks.

B. Key Observations in Inference Tasks

In collaborative inference tasks, like user interactions and video recognition in end devices, the data often exhibits



(a) Time locality visualization as a color line, in which similar colorations denote similar features.



(b) Spatial locality visualization as 2D points, color-coded by optimal quantization precision.

Fig. 1: Data correlation visualization on the UCF101 dataset with the ResNet101 model.

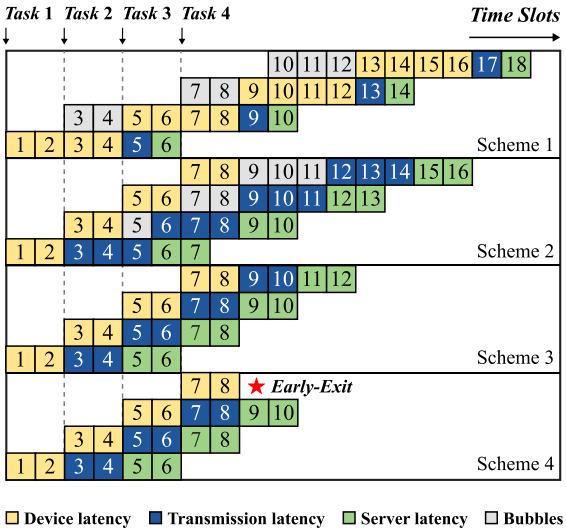


Fig. 2: Three-stage collaborative inference processes with pipeline scheduling.

repetitive patterns [29]. These patterns lead to the generation of similar intermediate data during the inference process, providing a solid foundation for predictive optimization in subsequent tasks. By leveraging the predictable nature of data correlations, substantial benefits can be achieved, including streamlined computations and efficient pipeline execution [30]. To highlight the importance of data correlation, we focus on both the temporal and spatial localities of intermediate data using the ResNet101 model [31] on the widely-used UCF101 video dataset [32]. Additionally, building on experiences in Section IV-B with the ImageNet-100 dataset [33], we demonstrate the versatility of our observations across various scenarios. We analyze the intermediate data across frames over extended periods, utilizing the t-SNE technique [34] to visualize the data features as a color line. Fig. 1(a) demonstrates pronounced consistency in the coloration of features over short intervals, indicating strong temporal locality among the intermediate data. This observed correlation supports the feasibility of temporarily caching previously computed results to accelerate future collaborative inference processes.

In Fig. 1(b), the data points are color-coded according to their optimal quantization precision (3-bit, 4-bit, or 5-bit), illustrating distinct clusters of inference tasks. Notably, data points with 3-bit precision typically cluster closer to the task center, whereas those with 5-bit precision are more widely dispersed. This observation indicates that data points more resistant to clustering necessitate higher quantization precision to preserve inference accuracy. The analysis of spatial locality reveals a clear correlation between task specificity and quantization efficiency, emphasizing the potential to enhance inference processing by customizing quantization precision according to the variability of task-specific features [29]. The predictable nature of data correlations, as demonstrated in Fig. 1, supports the implementation of an efficient early-exit policy and adaptive quantization adjustment. The above strategies enhance transmission efficiencies, contributing to bubble-free pipeline scheduling from a novel perspective.

C. Potential Opportunities of Collaborative Inference

The essence of collaborative inference lies in optimizing the utilization of heterogeneous computing resources across end devices and servers. Fig. 2 illustrates the three-stage collaborative inference processes, where four inference tasks arrive sequentially every 2 time units. Scheme 1 is designed to minimize latency per task, achieving a duration of 6 time units per task [35]. Though Scheme 1 employs pipeline parallelism, it does not fully optimize resource utilization, leading to numerous bubbles in pipeline execution. In contrast, Scheme 2 employs an alternative partitioning strategy specifically aimed at reducing bubbles in pipeline scheduling, which slightly increases task latency to 7 time units but significantly boosts system throughput. Scheme 2 effectively manages both transmission and computation stages, resulting in fewer bubbles and enhanced system efficiency [10]. This strategy demonstrates a practical advantage, achieving a 25% efficiency increase over Scheme 1, as the maximum stage is reduced from 4 to 3 time units in pipeline execution.

Building upon the strengths of Scheme 2, Scheme 3 introduces an adaptive quantization adjustment that capitalizes on intrinsic data correlations and task-specific features to further minimize pipeline bubbles. This enhancement, as explored in Section II-B, moves to achieve bubble-free pipeline execution and results in a reduction of 50% compared to Scheme 1, as the maximum stage is reduced from 4 to 2 time units. Furthermore, Scheme 4 integrates an early-exit policy that capitalizes on the temporal localities of inference tasks to streamline the inference processing.

In summary, as demonstrated in Fig. 2, these strategies in Schemes 2-4 show significant potential in minimizing pipeline bubbles during the inference process. The following sections will further explore meticulously designed strategies with offline and online components, aimed at realizing near bubble-

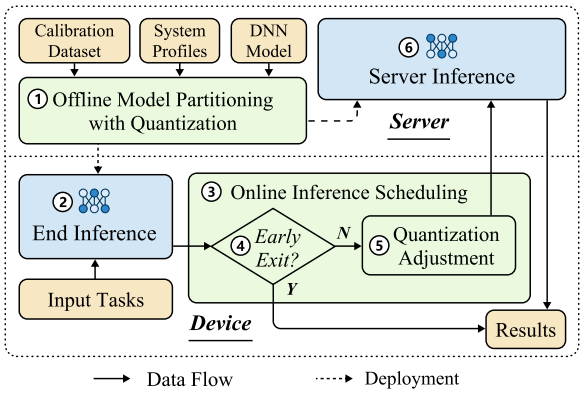


Fig. 3: Overview and inference workflow of COACH.

free pipeline execution and enhancing collaborative inference performance in terms of latency and throughput.

III. System Design of COACH

A. System Overview

To enhance the effectiveness of collaborative inference, we concentrate on refining the collaborative inference process and exploiting data correlations. We introduce COACH, a near bubble-free collaborative inference framework to handle the complexities of DNN models and dynamic networks, ensuring efficient resource utilization while maintaining high accuracy. With the combined strengths of the offline and online components, COACH concurrently manages computation and transmission stages, preserving a fluid pipeline flow and minimizing bubbles in pipeline scheduling.

In Fig. 3, COACH starts with an offline component that handles model partitioning and quantization strategy (①), acquiring system profiles and a calibration dataset. This component, which executes one-time before the inference process, carefully manages layers of parallel execution and initializes the online component without affecting inference performance. After that, the DNN model is deployed on both the end device and cloud server, setting the stage for collaborative inference execution with minimized pipeline bubbles. During the inference process, input tasks are initially processed through the partitioned model on the end device (②), generating intermediate data. Subsequently, the online scheduling component (③) evaluates the task-specific quantization requirements of the intermediate data, facilitating adaptive quantization adjustment to stabilize pipeline execution in dynamic networks. Once the quantization adjustment meets the early-exit condition (*e.g.*, exceeding the early-exit threshold), the procedure will promptly generate the result (④). Otherwise, COACH adjusts the quantization precision to further minimize pipeline bubbles before transferring the data to the cloud server (⑤). The cloud server conducts the remaining computations and returns the inference results to the end device (⑥), thereby completing the inference task.

B. Offline Model Partitioning and Quantization Component

We develop a recursive divide-and-conquer algorithm tailored for optimizing model partitioning and determining the quantization precision, as presented in Algorithm 1.

Problem Formulation. To optimize transmission efficiency, COACH utilizes the Uniform Affine Quantization (UAQ) technique [36] to compress intermediate data efficiently while preserving accuracy. This quantization facilitates streamlined data transmission, allowing the cloud server to subsequently dequantize the data and continue the inference process. Optimal quantization precision is determined through a dichotomous search [37], [38], based on the correlation between higher precision and improved inference accuracy. We define $Q(v_i)$ as the quantization approach for layer v_i in the DNN model, correlating with inference accuracy $Acc(\cdot)$ and constrained by accuracy loss limit of $\epsilon = 0.5\%$, thereby ensuring robust inference performance [4]:

$$|Acc(v_i) - Acc(Q(v_i))| \leq \epsilon \tag{1}$$

Let V_e and V_c denote the set of layers executed on the end device and cloud server, respectively, with V_p representing the partition layer set. The latency of three stages in the pipeline can be defined as:

$$T_e = \sum_{v_i \in V_e} t_i^e, \quad T_t = \sum_{v_i \in V_p} Q(t_i^t), \quad T_c = \sum_{v_i \in V_c} t_i^c, \tag{2}$$

where t_i^e, $Q(t_i^t)$, and t_i^c represent the computation time on the end device, the transmission time with quantization, and the computation time on the cloud for layer v_i, respectively. The task latency of collaborative inference is guaranteed in pipeline inference with $T_{\max}$ as:

$$T_e + T_t + T_c \leq T_{max}. \tag{3}$$

Utilizing the DAG topology, layer parallel execution enables the simultaneous processing of non-interdependent layers. As shown in Fig. 4, after end device computing of layer 3, subsequent end device computing (*e.g.*, layers 5 and 6) and transmission V_0^1 can execute in parallel. After completing transmission V_0^1, the transmission V_0^2 and cloud computing of layer 4 can execute in parallel, while cloud computing of layer 7 waits for transmission completion, introducing additional pipeline bubbles. By analyzing the layer dependencies, we carefully manage the layer parallel execution, considering the transmission and cloud parallel times T_t^p and T_c^p, respectively, with the following constraint:

$$T_t^p + T_c^p \leq \max\{T_e, T_t, T_c\}. \tag{4}$$

Furthermore, we introduce two bubble functions for pipeline scheduling evaluation: $B_c(V_p)$ for computation bubbles and $B_t(V_p)$ for transmission bubbles, defined as:

$$\begin{cases} B_c(V_p) & = |T_e - T_c|\,, \\ B_t(V_p) & = |T_t - \max\{T_e, T_t - T_t^p, T_c - T_c^p\}|\,. \end{cases} \tag{5}$$

The objective of the offline component is to identify an optimal partitioning and quantization strategy V^* that maximizes the overall pipeline efficiency by minimizing the key bottlenecks, including bubbles and the maximum latency. The

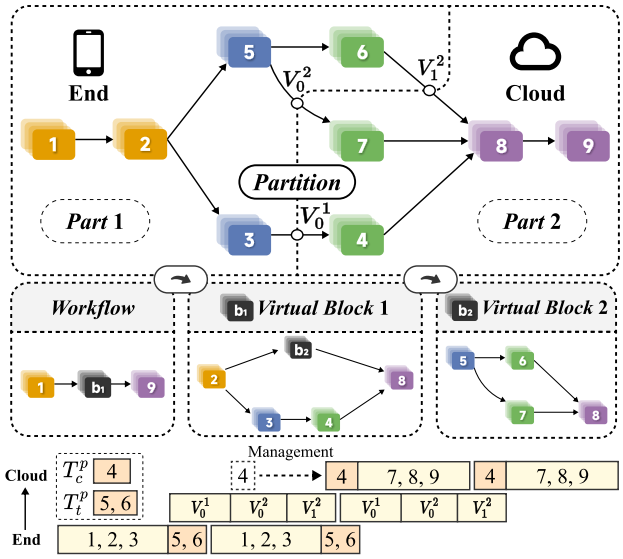


Fig. 4: Illustrating DNN partitioning with virtual blocks and layer parallel execution.

optimization problem for near bubble-free pipeline in COACH is formulated as follows:

$$V^* = \underset{V_p}{\operatorname{argmin}} \{B_c(V_p) + B_t(V_p) + \max\{T_e, T_t, T_c\}\} \quad s.t. \quad (1),\ (3),\ (4). \tag{6}$$

Algorithm Description. The principal challenge arises from the impracticality of exploring all possible partitions of a DAG, due to the exponential increase in the potential model partitioning search space as the model complexity grows. For chain topology, partition layer set V_p only comprises a single layer, whereas for DAG topology, V_p comprises multiple layers. We introduce a novel recursive divide-and-conquer algorithm to address the complexities of partitioning DAG topology models. As illustrated in Fig. 4, our algorithm is achieved by conceptually clustering parallel layers into virtual blocks and shifting the optimization focus towards a simpler chain topology, enabling carefully managed layer parallel execution (*e.g.*, layer 4) to ensure minimal pipeline bubbles.

In Algorithm 1, the offline component initiates with evaluating the computation and transmission costs of each layer in model G, which are pivotal for determining the latency of pipeline stages (Line 2). To simplify the exploration, the algorithm divides the complex DAG topology into more manageable chain topology segments, thus initiating a search space S for recursive analysis. COACH clusters parallel layers into virtual blocks within model G, organizing these blocks into a sequential chain flow B, as $\{1, b_1, 9\}$ in Fig. 4, which is incorporated in the search space as $S = \{B\}$ (Line 3).

The algorithm proceeds by iteratively evaluating each chain flow within the search space S, aiming to identify the corresponding optimal partitioning and quantization strategy V_B. For each chain flow B in search space S, the algorithm initially utilizes a dichotomous search to precisely determine the appropriate quantization precision for the inference data flow. Subsequently, by recognizing the dependency relationships among layers, it facilitates the analysis of parallel execution times T_t^p and T_c^p. This analysis enables the calculation of both the computation and transmission bubble functions $B_c(V_B)$ and $B_t(V_B)$, thereby the optimal strategy V_B is assessed based on the pipeline performance, as V_0^1, V_0^2 and V_1^2 in Fig. 4 (Lines 6-11). Within the chain flow B, the algorithm sequentially examines each virtual block, excluding those unsuitable for the strategy from further analysis. Conversely, suitable virtual blocks are recursively integrated into the search space S for optimization, ensuring thorough and efficient exploration of potential partitions, as b_1 and b_2 in Fig. 4. Following the comprehensive evaluation of B with layer parallel execution management, the offline strategy V^* is updated with V_B (Lines 12-15). After thoroughly exploring the search space F, the algorithm finalizes the most effective partitioning and quantization strategy V^*, as the strategy $\{V_0^1, V_0^2, V_1^2\}$ in Fig. 4. For a DAG model composed of n parallel data

Algorithm 1: Offline and Online Components in COACH

1 **Offline Component:**
2 Evaluate layer computation and transmission costs in model G with system profiles P.
3 Cluster parallel layers into virtual blocks within G.
4 Determine chain flow $B_g = \{b_1, b_2, \ldots, b_n\}$ in G.
5 Initialize search space $S = \{B_g\}$ and offline strategy $V_p = \emptyset$.
6 **for** *each chain flow* B *in* S **do**
7 Initial offline strategy $V_B = \emptyset$ for B.
8 **for** *each block* b *in* B **do**
9 Determine quantization precision for block b.
10 Calculate bubble functions $B_c(V_B)$, $B_t(V_B)$.
11 Assess V_B performance in pipeline execution.
12 Update the optimal offline strategy V_B for chain flow B.
13 **if** b *is a virtual block* **then**
14 Append additional chain flows from b to S.
15 Update V_p with V_B.
16 Deploy the optimal strategy V^* to device and server.
17 **Online Component:**
18 Initialize semantic centers $\mathbf{T_c}$ by calibration dataset D.
19 Evaluate early-exit threshold S_{ext} and quantization thresholds $\mathbf{S}_{adj}$.
20 **while** *Receive task feature* F **do**
21 Get similarity degrees $\mathbf{T}$ and task separability S.
22 **if** $S > S_{ext}$ **then**
23 Update $T_j^c \in \mathbf{T_c}$ by Eq. (7).
24 Get inference result R from $\mathbf{T_c}$ by Eq. (10).
25 Continue.
26 Get real-time network bandwidth and estimate quantization requirement Q_r with $\mathbf{S}_{adj}$.
27 Quantize F with Q_c by Eq. (11)

flows, each containing c independent layers, the conventional approach exhibits a time complexity of $O(c^n)$ [39]. Our algorithm achieves a substantially reduced time complexity of $O(cn)$, illustrating a significant improvement in efficiency for optimizing complex DNN models.

C. Online Inference Scheduling Component

The offline model partitioning and quantization strategy might not sufficiently ensure bubble-free pipeline execution. Additionally, dynamic network conditions significantly impact data transmission, potentially introducing bubbles in pipeline scheduling [40]. To address this, we introduce an online inference scheduling component that dynamically adjusts the quantization precision and implements an early-exit policy, thereby stabilizing pipeline execution in dynamic networks. Crucially, different tasks require varied quantization precision to maintain inference accuracy, facilitating further optimization of pipeline scheduling, as detailed in Section II-B. To make efficient real-time decisions on quantization adjustments, we propose a context-aware caching strategy that maintains label semantic centers. This strategy allows for quantization adjustments to be tailored to specific tasks, as outlined in the online component of Algorithm 1.

Label Semantic Centers with Caching Mechanism. To efficiently manage large volumes of intermediate data, COACH utilizes the Global Average Pooling (GAP) function [41], which concentrates on the core characteristics of data. Applied to intermediate data of dimensions $\langle C \times H \times W \rangle$, the GAP function simplifies the data into a reduced dimension of $\langle C \rangle$, where C denotes the number of channels, and H and W represent the spatial dimensions. This process generates *task features* F for intermediate data, which are concise feature vectors encapsulating essential characteristics.

Moreover, to make similarity-based decisions for quantization adjustment, the caching mechanism maintains a *semantic center* for each label, denoted as $\mathbf{T_c} = \{T_j^c\}$, where label $j \in \{0, \ldots, n\}$. The offline strategy determines the quantization precision based on intermediate data after quantization, ensuring the accuracy of the inference process (*e.g.*, 0.5% accuracy loss). Moreover, by analyzing the spatial locality of task features, we observe that intermediate data clustering around the label semantic centers requires less information transmission to maintain accurate inference, allowing for more aggressive quantization strategies.

Evaluating the correlation between intermediate data and the label semantic centers in the cache allows for dynamic adjustment of quantization precision, accommodating network fluctuations and minimizing pipeline bubbles. These label semantic centers are initially warmed up with the calibration dataset D and are gradually updated with task features during the inference process. The label semantic centers remain a true reflection of current conditions by continuously integrating new task features, thereby reducing potential data biases. The label semantic center is updated as follows:

$$T_j^c = \frac{m_j T_j^c + F_j}{m_j + 1}, \tag{7}$$

where m_j denotes the count of tasks, T_j^c is the semantic center and F_j is the current task feature associated with label j.

Task Separability for Quantization Adjustment. To evaluate correlations between intermediate data and label semantic centers for precise quantization adjustment, we first introduce the concept of similarity degree between task features and label semantic centers. For assessing similarity, the cosine distance metric [42] is employed to compare task features with label semantic centers, establishing similarity degrees $\mathbf{T} = \{t_j\},\ j \in \{0, \ldots, n\}$ for each label. The similarity degree for label j is formalized as:

$$t_j = \xi(F, T_j^c) \in [0, 1], \tag{8}$$

where the function $\xi(\cdot) \in [0, 1]$ represents the cosine similarity function [43]. The term t_j quantifies the similarity degree between the task feature F and the label semantic center T_j^c, with $t_j = 1$ indicating perfect similarity and $t_j = 0$ indicating no similarity. Additionally, to effectively evaluate the quantization precision requirements for task-specific intermediate data, we introduce the concept of task separability, denoted as S:

$$S = \|\mathbf{T}\|_2 \cdot (t_H - t_{SH}) \frac{t_H}{t_{SH}}, \tag{9}$$

where t_H and t_{SH} represent the highest and second-highest degrees of similarity within the set $\mathbf{T}$, respectively. A higher value of S signifies a more pronounced correlation between the intermediate data and the semantic center of the label, suggesting that the inference results are more reliable for the target label. This reliability allows for more aggressive quantization, facilitating efficient transmission.

Context-Aware Acceleration Strategy. Utilizing the label semantic centers $\mathbf{T_c}$ and task separability S, COACH introduces a context-aware acceleration strategy that dynamically adjusts quantization precision and employs an early-exit policy. The early-exit threshold S_{ext} and quantization precisions thresholds $\mathbf{S}_{adj}$ are initially established using the calibration dataset D to ensure an accuracy loss below 0.5% [4], which only execute one-time prior to running. When task separability S exceeds the cache threshold S_{ext}, the conditions are met for an early return of inference results. The decision-making framework for the early-exit result R is encapsulated by the following equation:

$$R = \underset{j \in \{0, \ldots, n\}}{\operatorname{argmax}} \{t_j \in \mathbf{T}\}, \tag{10}$$

where the result is ascertained by identifying the highest similarity degree among all labels within $\mathbf{T}$. The early-exit policy ensures that decisions are predicated on the most relevant and similar data features, thereby optimizing operational efficiency while maintaining inference accuracy.

In addition, quantization adjustment plays a pivotal role in stabilizing pipeline execution within dynamic network environments. While offline model partitioning and quantization strategy establishes initial data precision parameters, further minimizing pipeline scheduling bubbles during inference is essential. This adjustment relies on the premise that greater

task separability S enables a more aggressive quantization approach, thus preserving the accuracy of inference results. By comparing the task separability S with quantization adjustment thresholds $\mathbf{S}_{adj}$, the quantization precision requirement Q_r for the current task is precisely determined. This precision is then applied to make a real-time decision on quantization adjustment Q_c by minimizing the bubble function:

$$Q_c = \underset{Q_c \geq Q_r}{\operatorname{argmin}} \{|T_t' - \max\{T_e, T_t', T_c\}|\}, \tag{11}$$

where T_t' denotes the transmission time with quantization precision Q_c and real-time bandwidth B. This formula facilitates task-specific quantization adjustments to further minimize pipeline bubbles, enhancing the efficiency of collaborative inference in dynamic network conditions.

IV. Performance Evaluation

A. Experimental Settings and Baselines

System Implementation. The experimental setup includes a high-performance AMAX deep learning workstation as the cloud server, which is equipped with an Intel Xeon Octa-core processor and 4 NVIDIA A6000 GPUs. To accurately simulate end devices with varying resources, our platform incorporates both a Nvidia Jetson Xavier NX and a Nvidia Jetson TX2. The NX model is equipped with a 6-core NVIDIA Carmel ARMv8.2 CPU and a 384-core NVIDIA Volta GPU, complemented by 8GB of RAM. Conversely, the TX2 features a 4-core ARM Cortex-A57 CPU and a 256-core NVIDIA Pascal GPU, also with 8GB of RAM. Furthermore, the network setup in our experimental platform includes a 5GHz WiFi router, enabling to establish wireless connections and simulate real-world network conditions. We maintain a strict accuracy loss threshold of 0.5%, ensuring consistent inference performance [4].

Datasets and Models. The datasets and models in our experiments are detailed as follows:

- **UCF101** [32] is a well-known video dataset primarily used for human action recognition research, comprising 13,320 short videos across 101 action categories. For our experiments, we select several frames per second (*e.g.*, 20 frames/sec) to construct continuous inference tasks, arranged chronologically to simulate the real-time activities observed in the videos.
- **ImageNet-100** [33] is a curated subset of the extensive ImageNet database, comprising 100 distinct object categories. To evaluate the nature scenes in mobile computing, we split and shuffle the ImageNet-100 dataset with long-tail distribution. This involves allocating a higher selection ratio for more common categories and a lower ratio for less frequent ones.

Our evaluations implement two widely-used DNN models, which achieve high accuracy for real-world applications.

- **VGG16** [12] is characterized by the utilization of 3×3 convolutional filters, allowing it to extract intricate features from images for tasks such as image classification and object recognition. It consists of 16 processing layers and has 121 million parameters, enhancing its robust performance in diverse applications.
- **ResNet101** [13] employs Residual Network architecture, which allows gradients to flow through a shortcut path during backpropagation, effectively addressing the vanishing gradient problem in deep networks. It features a deep network of 101 layers and incorporates 45 million parameters with DAG topology.

TABLE I: Average Inference Latency (ms) for COACH and baselines.

	Resnet101		**VGG16**	
	NX	**TX2**	**NX**	**TX2**
NS	45.16	62.67	29.52	52.73
DADS	38.11	49.34	24.38	39.50
SPINN	22.04	30.10	14.03	19.97
JPS	20.78	28.31	12.52	18.13
COACH	**15.63**	**19.11**	**9.71**	**13.37**

Baselines and Metrics. To comprehensively evaluate the inference performance, we compare COACH with the following four baselines:

- *Neurosurgeon (NS)* [6] optimizes the inference latency of individual tasks through the partition strategy of DNNs, specifically tailored for chain topology models.
- *DADS* [3] introduces a model partition strategy for pipeline execution designed to optimize performance in both lightly loaded and heavily loaded networks.
- *SPINN* [27] utilizes a dynamic partition strategy and fixed quantization compression, along with an early-exit mechanism to minimize latency under resource constraints.
- *JPS* [11] proposes a layer-level scheduling algorithm to achieve near-optimal pipeline scheduling for end device computation and transmission stages.

To conduct a comprehensive evaluation of COACH and baselines, we employ the following three performance metrics:

- Inference Latency (ms) measures the time required to process each task, critical for assessing the responsiveness of collaborative inference systems.
- Transmission Cost (Kb) evaluates the average transmission overhead for each task, providing insights into the efficiency of data transfer.
- System Throughput (it/s) represents the number of tasks the system can handle per second, providing a concrete measure of the processing capability and operational efficiency of a collaborative inference system.

B. Overall Performance

Our evaluations on latency and throughput are conducted across both high-performance (NX) and low-performance (TX2) devices, across network conditions ranging from 2Mbps to 100Mbps. We utilize the ImageNet-100 dataset to examine the performance of ResNet101 and VGG16.

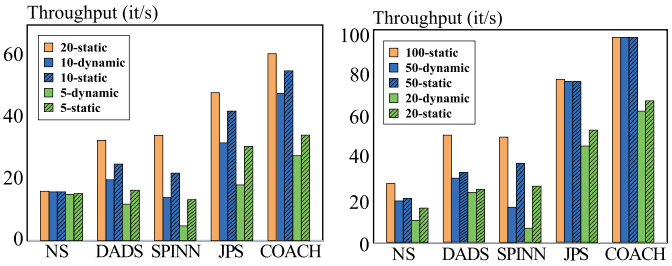
Latency Reduction of COACH. To assess the enhanced inference performance of COACH, we record the average inference latency for COACH and baselines. Table I illustrates that COACH consistently outperforms four baselines across all inference settings, achieving latency reduction from 22.48% to

TABLE II: COACH performance with context-aware acceleration across different data correlation levels.

	Resnet101			VGG16		
	EER	**Lat.**	**Trans.**	**EER**	**Lat.**	**Trans.**
NoAdjust	-	13.63	121.0	-	12.02	98.0
Low	11.86%	12.19	102.9	19.77%	10.64	75.3
Medium	37.41%	9.68	80.2	52.69%	7.71	44.1
High	65.38%	6.13	40.3	76.58%	5.77	20.6

73.59%. Notably, the latency reduction benefit of COACH is more pronounced in scenarios involving low-performance devices (TX2) and complex DNN models (ResNet101), indicating that COACH is effective in optimizing the complex environments of collaborative inference. Compared to NS, DADS, and SPINN, COACH demonstrates considerable inference improvements, with latency reductions of 69.17%, 61.26%, and 36.32% respectively. Moreover, despite the efficient performance of JPS, COACH still manages a 31.75% latency reduction, showcasing the effectiveness of its quantization strategy and context-aware acceleration. This performance is especially notable in reducing pipeline bubbles and enhancing computational resource utilization. The results underline the ability of COACH to deliver substantial latency improvements, affirming its effectiveness in facilitating lower latency.

Context-Aware Acceleration of COACH. Our analysis utilizes the UCF101 video dataset to evaluate the performance of context-aware acceleration in COACH across various data correlation levels, ranging from low (random frames), through medium (continuous frames from random videos), to high (continuous frames from sequential videos). The NoAdjust scenario, which does not employ context-aware acceleration, serves as a baseline for comparison. The results reveal a distinct correlation between data correlation levels and performance improvements, particularly quantified by early-exit ratio (EER), latency in ms (Lat.), and transmission costs in Kb (Trans.), as presented in Table II. Specifically, the transition from low to high data correlation levels results in early-exit ratio rising from 11.86% to 65.38%, with a significant reduction in latency (from 12.19 ms to 6.13 ms) and transmission costs (from 102.9 Kb to 40.3 Kb). Clearly, COACH is consistently effective at enhancing performance across various data correlation levels, underscoring its adaptability and potential across diverse scenarios. Compared to the NoAdjust scenario, incorporating context-aware acceleration within COACH leads to substantial performance enhancements. Notably, at high data correlation levels using ResNet101, there is an impressive 65.38% increase in the early-exit ratio, alongside a significant 66.70% reduction in average transmission costs and a 55.03% decrease in latency. These improvements are facilitated by the implementation of an adaptive quantization method to optimize the inference process. Similar benefits are observed with VGG16, which demonstrates the extensive advantages of the early-exit policy and quantization adjustment implemented by COACH under conditions of high data correlation. These enhancements underline the efficacy of the context-aware acceleration strategy in optimizing inference processes, particularly by further minimizing pipeline bubbles in the inference process and reducing unnecessary data transmissions.



(a) Initial at 20Mbps, reduced to 10Mbps and 5Mbps

(b) Initial at 100Mbps, reduced to 50Mbps and 20Mbps

Fig. 5: Adaptability of COACH and baselines in dynamic network conditions.

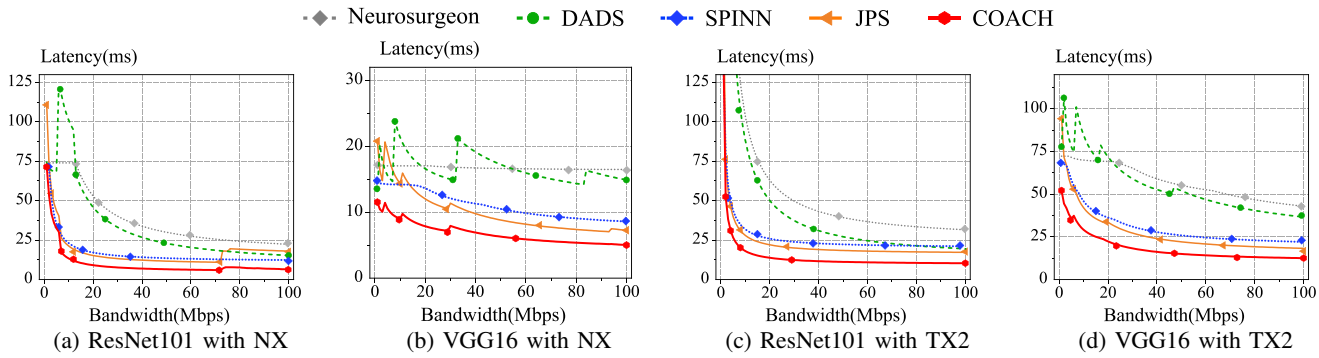
C. System Adaptability in Dynamic Networks

We assess the adaptability of COACH to dynamic network conditions through a series of experiments conducted on the ImageNet-100 dataset. These experiments allow for a detailed analysis of performance across network fluctuations, from initial bandwidths of 20Mbps decreasing to 10Mbps and further to 5Mbps in Fig. 5(a), as well as from initial bandwidths of 100Mbps decreasing to 50Mbps and 20Mbps in Fig. 5(b). Such conditions are known to potentially introduce numerous bubbles in pipeline execution. We define *static throughput* as the optimal throughput and *dynamic throughput* as the decreased throughput when bandwidth is reduced. COACH consistently outperforms all baselines with network fluctuations, showcasing enhanced throughput across all bandwidth settings. In Fig. 5(a), COACH initially surpasses the throughput of the leading JPS by 1.3×. When the bandwidth is reduced to 10Mbps, this advantage expands to 1.4× compared to JPS, with only a 12% reduction from its *static throughput* in this bandwidth setting. At a further reduced bandwidth of 5Mbps, the throughput of COACH still stands 1.6× higher than that of JPS, marking just a 15% decrease from the *static throughput*. These results underscore the robustness of COACH, maintaining high throughput consistency and minimizing pipeline bubbles. In Fig. 5(b), COACH maintains high throughput consistency, recording 95 it/s even when the bandwidth is halved to 50Mbps, staying 1.2× higher than JPS. This impressive performance showcases COACH can effectively mitigate the impacts of network fluctuations and reduce pipeline bubbles, highlighting its potential and broad prospects for real-world application.

D. Impact of Bandwidth on Collaborative Inference

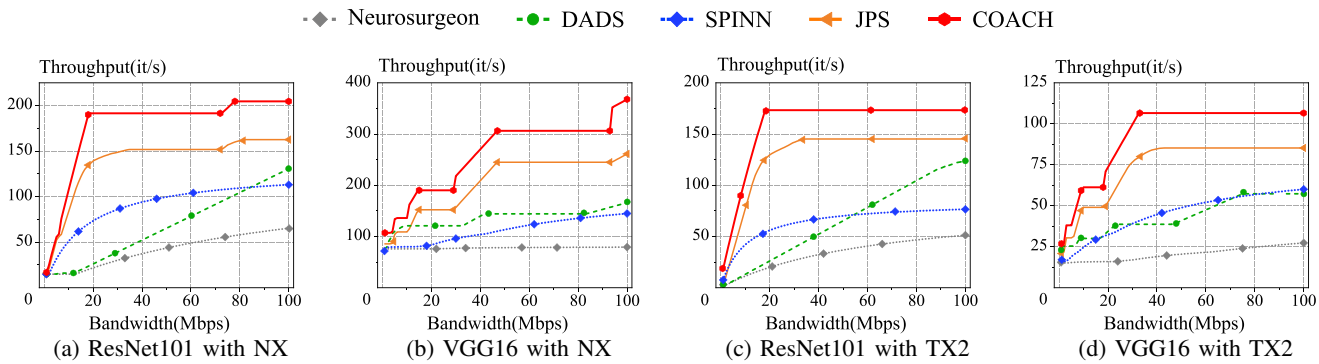
We conduct a comprehensive evaluation of COACH across diverse settings, leveraging the UCF101 dataset to simulate various network bandwidth scenarios ranging from 1Mbps to 100Mbps. The experiments are designed to test the system performance across both high-performance (NX) and low-performance (TX2) end devices.

Latency Performance. The experimental results on collaborative inference latency, as illustrated in Fig. 6 for ResNet101



(a) ResNet101 with NX (b) VGG16 with NX (c) ResNet101 with TX2 (d) VGG16 with TX2

Fig. 6: Latency of COACH and baselines on Resnet101 and VGG16 in different scenarios.



(a) ResNet101 with NX (b) VGG16 with NX (c) ResNet101 with TX2 (d) VGG16 with TX2

Fig. 7: Throughput of COACH and baselines on Resnet101 and VGG16 in different scenarios.

and VGG16, demonstrate the superiority of COACH in achieving the lowest latency across various bandwidth settings, markedly surpassing baselines. For ResNet101, under low bandwidth conditions, such as 10Mbps in Fig. 6(a), COACH achieves latency reductions of 72% and 38% compared to NS and JPS, respectively. In higher bandwidth scenarios (*e.g.*, 50Mbps), COACH reduces latency by 71% against NS and 42% against JPS. Similarly, for VGG16 in Fig. 6(b) and Fig. 6(d), COACH outperforms the baselines across all network conditions. COACH leverages cache acceleration and quantization adjustment to achieve latency improvements of 55% and 38% over NS and JPS in low bandwidth settings such as 5Mbps. In high bandwidth settings such as 70Mbps, COACH reduces latency by 72% and 40% compared to NS and JPS, respectively. This performance is attributed to the effective utilization of partitioning strategy and context-aware acceleration in COACH, which are optimized for varying network conditions. These results underline the robustness of COACH in adjusting dynamically to network variations, optimizing both inference performance and resource efficiency.

Throughput Performance. Fig. 7 illustrates the superior system throughput performance of COACH on ResNet101 and VGG16, underscoring its adaptability across a range of computational resources and network conditions. For ResNet101 in low bandwidth conditions, such as 10Mbps in Fig. 7(a), where transmission resources are the bottleneck of the inference process, COACH significantly enhances throughput, achieving increases of 6.2× compared to NS and 1.6× compared to JPS. In high bandwidth conditions such as 50Mbps, where computation resources are the bottleneck, COACH improves throughput by 4.5× over NS and 1.4× over JPS. This superior performance is due to COACH's effective context-aware quantization adjustment, which optimizes resource utilization and minimizes bottlenecks. For VGG16, as shown in Fig. 7(b), COACH significantly enhances throughput, achieving a 9.3× increase over NS, 3.3× over SPINN, and 1.8× over JPS. These improvements are attributed to the well-optimized cooperation between the offline and online components in COACH, which ensures efficient pipeline scheduling.

V. Conclusion

In this paper, we propose COACH, a novel framework designed to minimize pipeline bubbles, thereby enhancing the efficiency of collaborative inference systems. We propose an offline component that incorporates an efficient optimization algorithm for enabling more efficient partition and quantization for pipeline execution. Furthermore, our online component employs a context-aware acceleration strategy to further reduce pipeline bubbles. Comprehensive experiments corroborate the superior performance of COACH, showing significant improvements in latency and throughput.

VI. Acknowledgement

This article is supported in part by the National Science Foundation of China (NSFC) under Grants 61936015 and 62132019; in part by the Jiangsu Province Science Foundation for Youths (Grant No. BK20230275); in part by the Anhui Province Science Foundation for Youths (Grant No. 2408085QF185); in part by USTC Research Funds of the Double First-Class Initiative (Grants No. WK2150110033 and No. WK2150110030).

REFERENCES

[1] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *2010 24th IEEE international conference on advanced information networking and applications*. Ieee, 2010, pp. 27–33.

[2] Y. Xu, Y. Liao, H. Xu, Z. Ma, L. Wang, and J. Liu, "Adaptive control of local updating and model compression for efficient federated learning," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5675–5689, 2022.

[3] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1423–1431.

[4] M. Almeida, S. Laskaridis, S. I. Venieris, I. Leontiadis, and N. D. Lane, "Dyno: Dynamic onloading of deep neural networks from cloud to device," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 6, pp. 1–24, 2022.

[5] S. Yang, Z. Zhang, C. Zhao, X. Song, S. Guo, and H. Li, "Cnnpc: End-edge-cloud collaborative cnn inference with joint model partition and compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4039–4056, 2022.

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

[7] S. Sonko, E. A. Etukudoh, K. I. Ibekwe, V. I. Ilojianya, and C. D. Daudu, "A comprehensive review of embedded systems in autonomous vehicles: Trends, challenges, and future directions," *World Journal of Advanced Research and Reviews*, vol. 21, no. 1, pp. 2009–2020, 2024.

[8] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *2018 IEEE 24th international conference on parallel and distributed systems (ICPADS)*. IEEE, 2018, pp. 671–678.

[9] J. Liu, J. Yan, H. Xu, Z. Wang, J. Huang, and Y. Xu, "Finch: Enhancing federated learning with hierarchical neural architecture search," *IEEE Transactions on Mobile Computing*, 2023.

[10] P. Qi, X. Wan, G. Huang, and M. Lin, "Zero bubble (almost) pipeline parallelism," in *The Twelfth International Conference on Learning Representations*, 2024.

[11] Y. Duan and J. Wu, "Optimizing job offloading schedule for collaborative dnn inference," *IEEE Transactions on Mobile Computing*, 2023.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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

[14] J. Yao, S. Zhang, Y. Yao, F. Wang, J. Ma, J. Zhang, Y. Chu, L. Ji, K. Jia, T. Shen *et al.*, "Edge-cloud polarization and collaboration: A comprehensive survey for ai," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6866–6886, 2022.

[15] Y. Liao, Y. Xu, H. Xu, Z. Yao, L. Wang, and C. Qiao, "Accelerating federated learning with data and model parallelism in edge computing," *IEEE/ACM Transactions on Networking*, 2023.

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

[17] W. Ren, Y. Qu, C. Dong, Y. Jing, H. Sun, Q. Wu, and S. Guo, "A survey on collaborative dnn inference for edge intelligence," *arXiv preprint arXiv:2207.07812*, 2022.

[18] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, and Y. Zhang, "Auto-split: A general framework of collaborative edge-cloud ai," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2543–2553.

[19] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism," *IEEE Transactions on Mobile Computing*, 2021.

[20] Y. Duan and J. Wu, "Computation offloading scheduling for deep neural network inference in mobile computing," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–10.

[21] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "Ionn: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proceedings of the ACM symposium on cloud computing*, 2018, pp. 401–411.

[22] L. Yang, C. Zheng, X. Shen, and G. Xie, "Ofpcnn: On-demand fine-grained partitioning for cnn inference acceleration in heterogeneous devices," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[23] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2019.

[24] J. Liu, J. Liu, H. Xu, Y. Liao, Z. Wang, and Q. Ma, "Yoga: Adaptive layer-wise model aggregation for decentralized federated learning," *IEEE/ACM Transactions on Networking*, 2023.

[25] W. Shi, Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng, "Improving device-edge cooperative inference of deep learning via 2-step pruning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 1–6.

[26] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon, "Clio: Enabling automatic compilation of deep learning pipelines across iot and cloud," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–12.

[27] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the 26th annual international conference on mobile computing and networking*, 2020, pp. 1–15.

[28] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[29] Y. Li, C. Zhang, S. Han, L. L. Zhang, B. Yin, Y. Liu, and M. Xu, "Boosting mobile cnn inference through semantic memory," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 2362–2371.

[30] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[31] J. Liu, S. Wang, H. Xu, Y. Xu, Y. Liao, J. Huang, and H. Huang, "Federated learning with experience-driven model migration in heterogeneous edge networks," *IEEE/ACM Transactions on Networking*, 2024.

[32] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[34] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[35] I. V. Bajić, W. Lin, and Y. Tian, "Collaborative intelligence: Challenges and opportunities," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 8493–8497.

[36] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[37] V. Y. Tsvetkov, "Dichotomous systemic analysis," *Life Science Journal*, vol. 11, no. 6, pp. 586–590, 2014.

[38] J. Yan, J. Liu, H. Xu, Z. Wang, and C. Qiao, "Peaches: Personalized federated learning with neural architecture search in edge computing," *IEEE Transactions on Mobile Computing*, 2024.

[39] Z. Xu, H. Peng, and W. Wang, "Ago: Boosting mobile ai inference performance by removing constraints on graph optimization," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[40] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 854–863.

[41] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[42] Z. Yao, J. Liu, H. Xu, L. Wang, C. Qian, and Y. Liao, "Ferrari: A personalized federated learning framework for heterogeneous edge clients," *IEEE Transactions on Mobile Computing*, 2024.

[43] H. V. Nguyen and L. Bai, "Cosine similarity metric learning for face verification," in *Asian conference on computer vision*. Springer, 2010, pp. 709–720.