A Fine-Grained End-to-End Latency Optimization Framework for Wireless Collaborative Inference

Lei Mu, Zhonghui Li, Wei Xiao, Ruilin Zhang, Peng Wang, Tao Liu, Geyong Min, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Mobile devices are becoming increasingly capable of delivering intelligent services by leveraging deep learning architectures such as deep neural networks (DNNs). However, due to the compute-intensive nature of these tasks, mobile devices often struggle to handle them independently, leading to the exploration of collaborative inference as a promising solution for achieving low-latency mobile intelligence. Despite its potential benefits, many challenges need to be addressed in realizing the full potential of inference acceleration. This article presents a collaborative device-edge inference optimization framework as a promising solution to inference acceleration. The framework comprises fundamental modules, including the parameters generator (PG), accuracy predictor (AP), delay calculator (DC), and optimizer (OP), which are specifically designed to identify the optimal set of parameters for model compression, DNN partition, and feature compression. To illustrate its implementation, an example of a deep CNN network is introduced, and the collaborative inference latency optimization is formulated as a mixed-integer programming problem. The implementation of a specific algorithm instance using a quantum-inspired OP within the optimization framework is then presented. A multiple regression-based inference accuracy prediction model is proposed to maintain inference accuracy close to that of the original network while significantly reducing the time consumption during the offline phase. Through various simulation scenarios involving inference tasks of AlexNet and RegNet on CIFAR-10, incorporating diverse hardware computation specifications and wireless communication link conditions, the proposed framework demonstrates superior performance in terms of inference acceleration compared to the compared methods.**

***Index Terms*—Collaborative inference, edge intelligence, latency optimization.**

Manuscript received 28 April 2023; revised 8 July 2023 and 4 August 2023; accepted 20 August 2023. Date of publication 23 August 2023; date of current version 6 February 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62171390, and in part by the Sichuan Science and Technology Program under Grant 2023YFG0302. The work of Lei Mu was supported by the China Scholarship Council and was performed during Lei Mu's visit in King's College London. *(Corresponding author: Lei Mu.)*

Lei Mu, Peng Wang, and Tao Liu are with the School of Computer Science and Engineering, Southwest Minzu University, Chengdu 610225, China (e-mail: truemoller@outlook.com; wp002005@163.com; tao_liu@swun.edu.cn).

Zhonghui Li is with the School of Intelligent Engineering, Hubei Enshi College, Enshi 445000, China (e-mail: li1819349545@gmail.com).

Wei Xiao is with the Faculty of Informatics, Eötvös Loránd University, 1053 Budapest, Hungary (e-mail: xw46604818@outlook.com).

Ruilin Zhang is with the School of Business, The University of Auckland, Auckland 1142, New Zealand (e-mail: rzha377@aucklanduni.ac.nz).

Geyong Min is with the Department of Computer Science, University of Exeter, EX4 4SB Exeter, U.K. (e-mail: g.min@exeter.ac.uk).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/JIOT.2023.3307820

I. Introduction

In RECENT years, deep-learning architectures, such as deep neural networks (DNNs) [1], have been applied to fields, including computer vision, speech recognition, and natural language processing. These models have produced results comparable to, and in some cases surpassing, human expert performance [2], [3], [4], [5]. However, as mobile smart devices gain popularity and Internet of Things (IoT) technology continues to grow, providing intelligent services on mobile devices faces both opportunities and challenges. DNN tasks that are increasingly complex and compute-intensive consume significant computing resources, beyond the capabilities of today's mobile devices, including device-only inference. To tackle this issue, mobile edge computing (MEC) [6] offers new possibilities for achieving low-latency mobile intelligence. By deploying DNN models on edge servers with abundant computational resources, mobile devices can offload raw data and accelerate inference on the server. However, this paradigm cannot support intelligent inference with low-latency requirements due to significant communication overhead incurred in data transmission.

Collaborative inference, which offers a compromise between the device-only and server-only approaches, has drawn a lot of attention, with significant research work being investigated [7], [8], [9], [10], [11]. In real-world applications, collaborative inference is widely utilized in scenarios involving resource-constrained devices. Two prominent scenarios are as follows.

1) *Privacy-Conscious Applications:* In certain medical contexts, ethical or regulatory concerns prohibit the transmission of patient data for inference outside the premises [12]. Likewise, in home security systems using smart cameras, users are cautious about uploading their raw images to servers due to privacy concerns. Collaborative inference facilitates the transmission of privacy-independent intermediate feature data instead of the original data [13].
2) *Latency-Sensitive Applications:* Computation-intensive and latency-sensitive tasks, such as autonomous driving, cyber-physical control systems, and robotics, necessitate swift processing of incoming data. Resource-constrained devices rely on powerful server computing resources, but the substantial throughput and limited bandwidth present challenges in latency management. Collaborative inference addresses these issues by reducing the volume of data transmitted to the cloud,

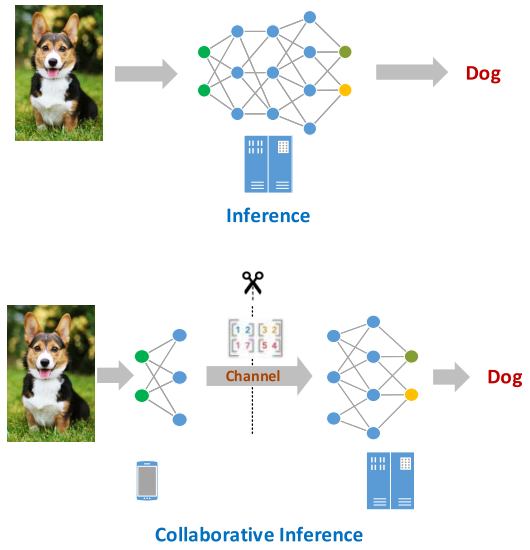


Fig. 1. Basic idea of collaborative inference.

thereby alleviating network traffic load and reducing latency [14].

In DNNs, it is common for the data size of certain intermediate layers to be significantly smaller than the raw input data. As demonstrated in the case of AlexNet [3], the input size undergoes a substantial reduction of approximately 75% after passing through the intermediate layer pool5 [7]. This presents an opportunity to accelerate inference by leveraging the powerful computational capacity of cloud computing and reducing communication latency, thereby optimizing both computation and communication delays.

The basic idea of collaborative inference is to split a DNN model at a cut layer, that is, a partition point, into a device-side model running on the device and a server-side model running on the edge server. The basic concept of collaborative inference is shown in Fig. 1.

As illustrated in Fig. 1, the device executes the on-device model with local data and sends the intermediate output associated with the cut layer to the edge server. Then the edge server received an intermediate feature vector as input of the server DNN partition for further processing and feeds back the inference result to the device. Selecting appropriate partitioning points enables the transmission of smaller intermediate feature data, leading to reduced communication latency. Furthermore, offloading a portion of the inference task to high-performance servers significantly decreases computational latency. Therefore, the partitioning strategy is essential for optimizing the inference latency in collaborative inference [7].

The fault tolerance property of neural networks [15] can be leveraged to reduce delay in collaborative inference. Model compression and feature compression are two prominent techniques for accelerating co-inference [16].

Various experiments are conducted with different model compression and feature compression parameters. The L1 Norm Pruner [17] in the neural network intelligence (NNI) [18] toolkit is employed to perform model pruning on the second convolutional layer of AlexNet. The results demonstrate that the model's accuracy decreases by only 2% with a 74% parameter pruning and by 5% with an 89% parameter pruning. The first fully connected layer of the AlexNet network is subjected to intermediate feature compression using the linear encoder as shown in Fig. 3. It is also observed that feature compression with a compression ratio of 6.5% results in an accuracy loss of no greater than 2%, while a compression ratio of 0.2% leads to an accuracy loss of approximately 7%.

These results indicate that model compression and feature compression are highly competitive methods to accelerate collaborative inference without significantly compromising model accuracy.

However, there are several challenges in the literature.

1) Fine-grained pruning with layerwise sparsity ratio can yield powerful model compression with allowable accuracy degradation [19], in contrast to coarsely characterizing pruning operations across the entire DNN model in previous studies. This highlights the potential for improving collaborative inference.
2) Optimizing the latency of collaborative inference poses challenges due to the expanded search space resulting from layer granularity sparsity selection and the exploration of various hyperparameter combinations for the encoder and decoder. Additionally, determining these optimal hyperparameters further complicates the optimization process.
3) Optimizing inference latency using these methods can negatively impact inference accuracy, making it crucial to evaluate whether the optimized DNN model meets accuracy requirements. However, due to the extensive search space and frequent parameter adjustments, the verification process can be time consuming.

To address these issues, a fine-grained and flexible optimization framework for collaborative inference is proposed to explore potential improvements in inference acceleration. The optimization problem is formulated and a heuristic algorithm is proposed to solve it. To mitigate the extensive verification work, an inference accuracy prediction model using multiple regression is proposed. The effectiveness of the framework is demonstrated by implementing it on the inference tasks of different models.

Compared with the existing work, the novelty of our framework is summarized in the following aspects.

1) A fine-grained end-to-end wireless collaborative inference framework is proposed to further reduce inference latency via model compression, DNN partition, and feature compression. The proposed framework is applicable to diverse end-to-end collaborative inference acceleration scenarios. It acts as a guiding tool for designing tailored collaborative inference acceleration algorithms. It is not a single algorithm, but rather an algorithm family, enabling users to select different techniques within the framework to create algorithm instances and automatically generate optimized strategies.
2) The optimization problem for collaborative inference latency is formulated as a mixed integer programming problem, where the decision variables correspond to

the layerwise sparsity ratios, the partition point, and encoder–decoder parameters. Subsequently, the implementation of a specific algorithm instance using a quantum-inspired OP showcases the feasibility and rationality of the proposed framework.

3) A multiple regression-based inference accuracy prediction model is proposed to predict inference accuracy and quickly evaluate if the candidate hyperparameter set meets accuracy requirements. This approach maintains inference accuracy while reducing time consumption in the offline phase and can be applied to other relevant scenarios.
4) Software simulations are conducted to evaluate the proposed collaborative inference optimization framework on the inference tasks involving AlexNet and RegNet. The simulation results demonstrate that the proposed optimization framework achieves significant inference acceleration compared to other methods.

The remainder of this article is organized as follows. In Section II, existing studies on collaborative inference acceleration in recent years are analyzed. Section III introduces the concept of collaborative inference between device and edge. The collaborative inference latency optimization framework is elaborated in Section IV, followed by simulation results in Section VI. Section VII concludes this article.

II. Related Works

This section provides an extensive survey of the literature and research conducted in the domain of collaborative inference acceleration, with a specific focus on DNN partitioning and model and feature compression.

A. DNN Partitioning

Given the potential for significantly smaller output from intermediate layers in a DNN model compared to the raw input data, different partitioning points have a direct impact on the on-device computational cost and communication overhead.

Neurosurgeon [7] is a lightweight scheduler that automatically partitions DNN computations between mobile devices and data centers at the granularity of neural network layers, leveraging the processing power of both mobile and cloud while minimizing data transfer overhead. Edgent [8] accelerates the DNN execution by optimizing partitioning and right-sizing, which enables early exiting and improves inference speed. DeepThings [9] is designed for the distributed execution of CNN-based inference applications on strictly resource-constrained edge servers, which takes into consideration the dynamic workload distribution and balancing at inference run time. DNNOff [10] employs a specialized program structure that facilitates on-demand offload, distributes computation across multiple devices, and dynamically decides on the offloading scheme. Liu et al. [11] proposed an adaptive framework to partition DNN computing among end devices, edge servers, and cloud servers based on layer prediction results for minimizing DNN inference latency.

B. Model and Feature Compression

DNNs have redundancy [20], which can be exploited to reduce computation time on local devices using model compression techniques, such as pruning, training quantization, and knowledge distillation [20]. Feature compression techniques like Huffman coding and neural network encoders [21], [22], [23] can reduce transmission time by encoding and compressing intermediate data. By reducing the data size of wireless transmission, they can significantly decrease inference latency.

JALAD [21] utilizes Huffman coding to compress and quantize the transmitted data features, and it designs an adaptive strategy for an edge-cloud structure that can dynamically change decoupling for different network conditions. BottleNet [24] introduces a bottleneck unit in a neural network, which significantly reduces the communication costs of feature transfer between mobile and cloud. A two-step feature coding approach [16] is proposed to reduce the transmission delay by data dimensional reduction and learning-driven coding. It can further compress the features by mapping symbols to code words using neural networks. Matsubara and Levorato [25] introduced a bottleneck in the early stages of DNNs for resource-constrained devices. The output is quantized and transmitted to the edge server, reducing the processing latency. Zhang et al. [26] endeavored to propose an automated machine learning (AutoML) framework to optimize communication-computation efficiency in device-edge co-inference by determining hyperparameters, such as model sparsity and feature compression ratio.

III. Collaborative Inference Acceleration Between Device and Edge

This section first presents the concept of collaborative device-edge inference acceleration and then introduces one instance for understanding its implementation.

A. Concept

To achieve collaborative inference acceleration, three primary steps are involved: 1) model compression; 2) DNN partitioning; and 3) feature compression. First, model compression reduces the computation latency of a pretrained DNN model by pruning the model to decrease its parameter count and inference cost. Next, DNN partitioning splits the modified DNN model into two parts by selecting a partition point, with one part deployed on the device and the other on the server. Finally, feature compression further reduces communication overhead.

These three steps effectively accelerate collaborative inference and have wide applicability to various inference acceleration scenarios for resource-constrained devices [7], [8], [9], [10], [11], [21], [22], [23], [24], [25], [26]. The concept is highly adaptable, with various techniques available for each step, enhancing its versatility. Specific techniques will be selected in the following paragraphs to provide clear illustrations and facilitate a better understanding of each step.

To achieve fine granularity of compression, we employ a layerwise pruning method that enables independent adjustment

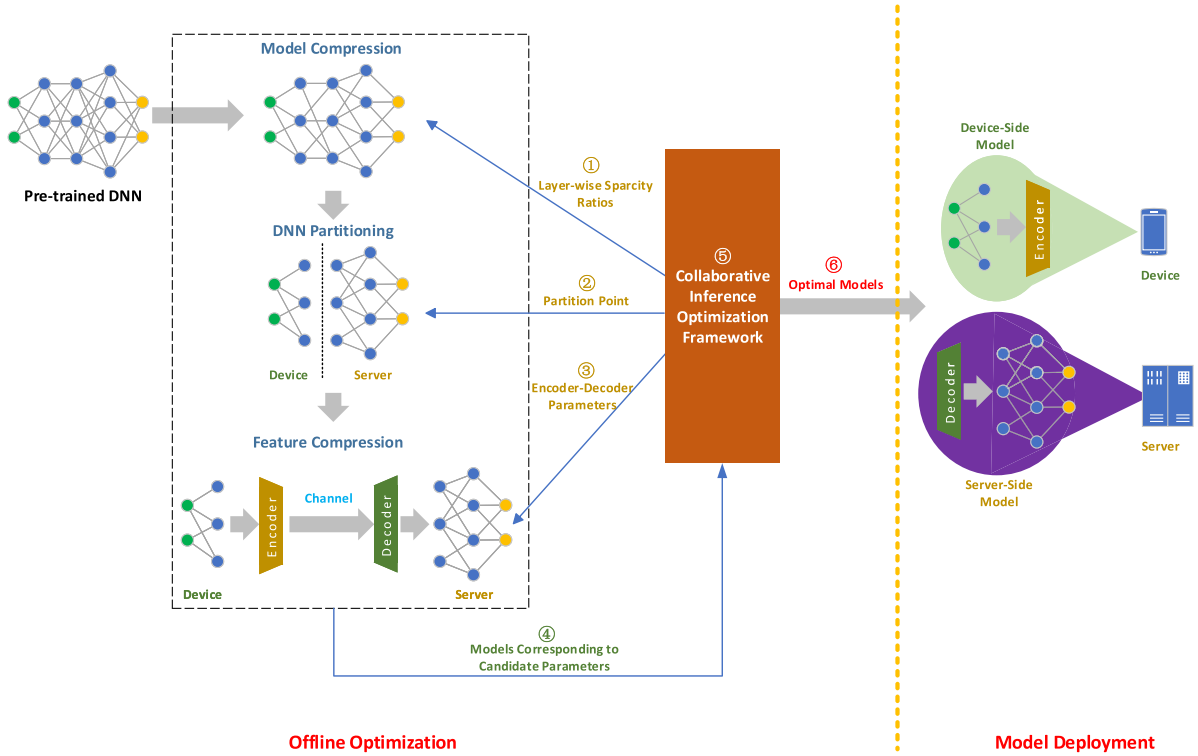


Fig. 2. Concept of the proposed collaborative inference process.

of sparsity ratios for each layer. Additionally, we implement feature compression by incorporating a lightweight complementary encoder and decoder pair into the two models. These techniques contribute to effective compression and reduction of computational overhead.

The entire collaborative inference acceleration process consists of two stages, namely, offline optimization and model deployment, as depicted in Fig. 2.

During the offline optimization stage, the proposed collaborative inference optimization framework achieves inference acceleration optimization through six steps.

1) Generating layerwise sparsity ratios for model compression.
2) Selecting a suitable partition point to divide the pre-trained DNN model into two parts.
3) Generating the parameters for a pair of complementary encoder and decoder, which are added to the different parts of the DNN model.
4) Constructing a DNN model corresponding to these parameters.
5) Fine-tuning and evaluating the candidate models based on various metrics, including latency and inference accuracy.
6) Outputting the optimal models after multiple rounds of iterative optimization for use in the next stage.

In the model deployment stage, the optimal models are deployed to the device and the edge server for collaborative inference acceleration.

B. Detailed Procedures

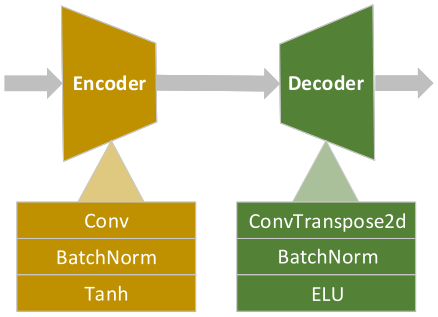
The L1 Norm Pruner [17] in the NNI [18] toolkit can be used to achieve fine-grained model compression by removing redundant parameters in the convolutional and fully connected layers of a deep CNN network like AlexNet. Each layer can have a separate sparsity ratio to achieve optimal compression results.

Feature compression reduces transmitted data, lowering latency and enhancing inference efficiency. Additionally, it bolsters privacy and security by limiting the information that can be inferred from compressed features. Once a partition point is chosen, the original DNN model splits into two parts, where the device-side model is appended with a lightweight encoder, and the server-side model is appended with a decoder, maintaining symmetry.

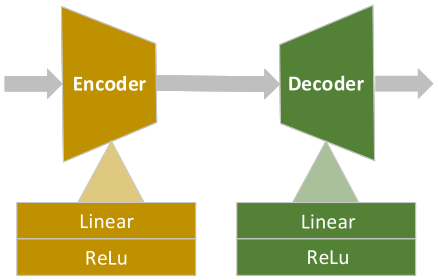
The fault-tolerant characteristics of DNNs [15] enable them to maintain low-accuracy loss through training even when some information is lost during transmission. To further minimize communication overhead and improve performance, joint source-channel coding in a task-oriented manner [16] is adopted in our approach. The goal is to achieve acceptable accuracy of the DNN model while reducing latency for a specific task, rather than enhancing message restoration at the receiver in a communication-oriented manner.

The proposed framework incorporates two types of encoders and decoders, designed to accommodate the varying output tensor dimensions of intermediate features resulting from different partition points in the DNN model. These trainable CNN structures are implemented using simple neural networks and trained end-to-end to maintain inference accuracy, compressing intermediate data through the convolutional and fully connected layer.

Based on extensive experiments, two encoder structures for different partition points of the neural network have been determined, as depicted in Fig. 3.



(a)



(b)

Fig. 3. Encoder and decoder structures for different partition points. (a) Convolutional encoder and decoder. (b) Linear encoder and decoder.

The proposed framework employs a multistep tuning training scheme to mitigate accuracy degradation caused by model and feature compression. The scheme includes several steps for fine-tuning the compressed model to improve accuracy.

1) The pretrained DNN network undergoes pruning using the NNI module, assigning sparsity ratios to different layers.
2) The resulting pruned model is fine-tuned to maintain model accuracy.
3) The modified DNN model is split into two parts by selecting a partition point.
4) Finally, the encoder and decoder are added to the two parts for joint source-channel coding by end-to-end training.

Following these offline training procedures, the model can maintain a relatively low level of accuracy degradation.

The selection of appropriate parameters to accelerate inference is a crucial issue, considering the wide range of parameter definitions and the numerous hyperparameter combinations.

IV. Proposed Collaborative Inference Latency Optimization Problem

In this section, we introduce a communication model and perform a layerwise delay analysis, laying the foundation for optimizing latency in collaborative inference. Subsequently, we conduct a detailed investigation into fine-grained end-to-end latency optimization for wireless collaborative inference.

TABLE I
Calculation of FLOPs and Output Data Size of DNN Layers

Layer Type	Γ (FLOPs)	D
Conv	$2H_oW_o(C_iK^2+1)C_o$	$H_oW_oC_o$
Relu	$H_iW_iC_i$	$H_iW_iC_i$
Maxpool	$K^2H_oW_oC_i$	$H_oW_oC_i$
Avgpool	$H_iW_iC_i$	$H_iW_iC_i$
Dropout	$H_iW_iC_i$	$H_iW_iC_i$
Fc	$(2C_i-1)C_o$	C_o

A. Communication Model

The communication latency of the inference results in the downlink is usually small compared to that of the uplink, so it is neglected in problem formulation for simplicity [27]. The communication model only considers the uplink and the device's transmission rate R can be calculated using the Shannon capacity formula [23]

$$R = B\log_2(1+\text{SNR}) \tag{1}$$

where SNR represents the signal-to-noise ratio (SNR) between the device and the server.

B. Layerwise Delay Analysis

The DNN model is composed of various types of layers with differing computational requirements and output data sizes, which in turn impact the computation and communication latencies.

The total latency of collaborative inference can be denoted as τ, which is the sum of computation latencies of the device and server, denoted by τ^d and τ^s, respectively, and the communication latency, denoted by τ^{tx}

$$\tau = \tau^d + \tau^s + \tau^{tx}. \tag{2}$$

The floating point operations (FLOPs) and output data size of layers in a DNN model are shown in Table I, where Γ and D represent the computation and output data size of the layer, respectively.

H_i, W_i, and C_i correspond to the height, width, and number of channels of the input intermediate feature, while H_o, W_o, and C_o correspond to the height, width, and number of channels of the output intermediate feature, respectively. The variable K represents the size of the convolution kernel used in the layer.

1) Computation Delay: The time delay of computation on the device can be calculated by

$$\begin{aligned}\tau^d &= \sum_{j=1}^{c}\tau_j^l \\ &= \sum_{j=1}^{c}\frac{\Gamma_j^M}{\Theta^d}\end{aligned} \tag{3}$$

where τ_j^l and Γ_j^M represent the time delay and the calculation after model compression for the jth layer. Θ^d is a measure of processor performance described by FLOPs per second (FLOPS).

The time delay of computation on the server can be calculated by

$$\tau^s = \sum_{j=c}^{N} \tau_j^l = \sum_{j=c}^{N} \frac{\Gamma_j^M}{\Theta^s} \tag{4}$$

where Θ^s is FLOPS on the server, and N is the number of layers in DNN model.

2) Communication Delay: The time delay of intermediate feature transmission is analyzed as follows:

$$\tau^{tx} = \frac{D_c^F}{R} \tag{5}$$

where D_c^F represents the output data size of the partition point c after feature compression.

C. Problem Formulation

The partition point, model compression strategy, and feature compression strategy are denoted by c, $\boldsymbol{S} = \{S(l) \mid l \le N_M, l \in \mathbb{N}_+\}$, and $\boldsymbol{P} = \{P(k) \mid k \le N_F, k \in \mathbb{N}_+\}$. The collaborative inference latency optimization can be established as

$$\begin{aligned} &\arg\min_{c,\boldsymbol{S},\boldsymbol{P}} \tau \\ &= \arg\min_{c,\boldsymbol{S},\boldsymbol{P}} \left(\tau^d + \tau^s + \tau^{tx}\right) \\ &= \arg\min_{c,\boldsymbol{S},\boldsymbol{P}} \left(\sum_{j=1}^{c} \frac{\Gamma_j^M}{\Theta^d} + \sum_{j=c}^{N} \frac{\Gamma_j^M}{\Theta^s} + \frac{D_c^F}{R}\right) \end{aligned} \tag{6}$$

$$\text{s.t.} \quad \begin{cases} 1 \le c \le N, \quad c \in \mathbb{N}_+ & (7) \\ 0 \le S(l) \le 1, l \le N_M, l \in \mathbb{N}_+ & (8) \\ P(k) \in \mathbb{N}_+, k \le N_F, k \in \mathbb{N}_+ & (9) \end{cases}$$

where $S(l)$ and $P(k)$ represent the lth sparsity ratio for layers in model compression and the kth parameter of encoder–decoder in feature compression. N_M and N_F are the number of parameters in model compression and feature compression. $S(l)$ and $P(k)$ have different expressions and value ranges if different model compression and feature compression methods are utilized.

Selecting appropriate parameters for DNN partition, model compression, and feature compression is crucial to achieving maximum inference acceleration. However, it is a complex and nonlinear mixed-integer programming problem involving numerous hyperparameters, such as layerwise sparsities and feature compression parameters, which exist in vast domains. To address this challenge, we deliver an offline collaborative inference framework capable of determining the optimal strategy.

V. Collaborative Inference Optimization Framework

This section illustrates the practical application of the framework to specific problem domains and outlines its fundamental modules. To enhance clarity and comprehension, specific techniques are chosen for each module and thoroughly explained, offering in-depth insights into the implementation details.

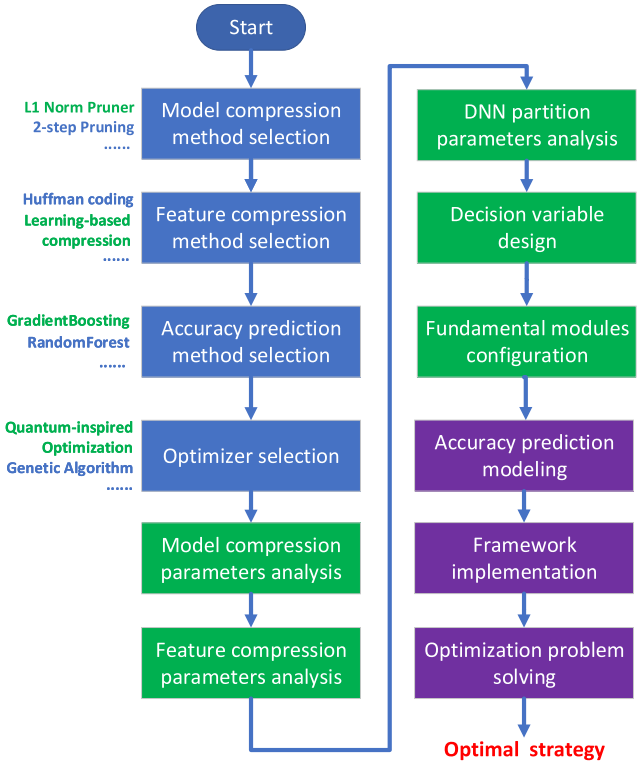


Fig. 4. Application workflow for proposed framework.

A. Framework Application Workflow

The application workflow for the proposed framework is shown in Fig. 4.

As shown in Fig. 4, the following steps should be followed to construct an instance that adheres to the framework when applying the framework to specific collaborative inference scenarios: technology architecture selection, framework configuration, and implementation.

The blue processes in the diagram represent the steps of technology architecture selection, where different techniques, such as model compression, feature compression, accuracy prediction, and OP methods, can be chosen to construct an instance to implement the framework. Different techniques can be selected and combined to create instances of algorithms within this framework. This implies that the framework corresponds to a family of algorithms formed by different algorithm instances.

The green processes in the diagram represent the steps of Framework Configuration. Once the technology architecture is determined, the decision variables that need to be optimized in each step, such as model compression, feature compression, and accuracy prediction, are analyzed systematically. Additionally, the fundamental modules within the framework are analyzed and configured accordingly.

The purple processes in the diagram represent the steps of Implementation. Before implementing the specific instance

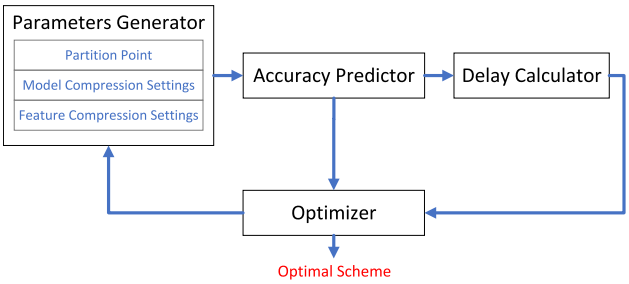


Fig. 5. Modules of the proposed collaborative inference optimization framework.

of the framework, the accuracy prediction model needs to be constructed based on the sampled data set using the selected method. Subsequently, the determined framework instance is progressively implemented step by step. The latency optimization problem for collaborative inference is solved using the selected OP, and the optimal strategy is obtained as the output.

B. Fundamental Modules

The fundamental modules of the collaborative inference optimization framework are described in Fig. 5.

The framework comprises four modules: 1) PG; 2) AP; 3) DC; and 4) OP.

1) Parameters Generator: PG in the Co-inference Architecture generates candidate strategies for decision variables, such as the partition point, model compression, and feature compression settings, based on specific requirements. These strategies serve as possible inputs for the other modules. The decision variables in our article include layerwise sparsity ratios, the partition point, and encoder–decoder settings. For AlexNet, there are seven parameters for layerwise sparsity ratios, one parameter for the partition point, and eight parameters for encoder–decoder settings. PG must guarantee that the generated parameters for the decision variables satisfy all the system constraints.

To ensure the effective processing of the candidate solution, it is crucial that each element operates within the same scale required by the framework. To address this issue, a mapping factor is introduced to calculate mapping values for the original elements, enabling the formation of a candidate solution in the same scale for iterative processing.

2) Accuracy Predictor: AP can significantly reduce the time needed to obtain the optimal decision variable strategy. It is no longer necessary to fully train and validate each DNN model corresponding to the generated parameters in PG as described in Section III-B. This greatly improves the efficiency of the collaborative inference framework.

AP enables the corresponding prediction accuracy to be obtained by inputting different decision variable strategies, without the need for conducting numerous experiments. After PG generates a set of candidate parameters that satisfy the system requirements, AP can calculate the predicted model accuracy for these parameters. If the accuracy requirements are met, the delay calculation can proceed in DC. Otherwise, the set of parameters will be discarded, and PG will generate a new set of candidates. This iterative process continues until a set of parameters that meets the accuracy requirements is found.

3) Delay Calculator: The complementary encoder and decoder pair used for feature compression is lightweight, and the computation introduced by them is negligible when compared to the entire network [23]. As a result, the total delay of the DNN model corresponding to the decision variable strategy can be calculated by DC using (2)–(5).

4) Optimizer: OP serves as the key component of the framework and aims to continuously optimize the decision variable strategy based on the results of the previous modules. To accomplish this, a quantum-inspired optimization algorithm [28] is utilized as the basis for OP.

The underlying theoretical concept of this framework is centered around the similarity between particle motion in quantum space and the search process in optimization space. This idea was initially proposed in Quantum annealing [29] and has been extensively recognized and investigated in various evolutionary algorithms [28], [30], including the quantum-inspired optimization algorithm in this framework. The optimal value of the objective function can be obtained when the quantum system converges to the ground state [31]. Therefore, the optimization problem can be considered as the search for the ground state in a quantum system with a constrained potential well. To solve this problem, the OP implemented in this framework consists of three main components: 1) energy level stabilization; 2) energy level transition; and 3) scale adjustment. Through multiple iterations, the OP will shrink to the optimal solution. The theory of the original quantum-inspired optimization algorithm [28] is beyond the scope of this work, and thus will not be discussed in detail.

The proposed optimization framework builds upon this OP, and further details will be elaborated on in the next section.

C. Implementation

The pseudocode of the proposed collaborative inference optimization framework is shown in Algorithm 1.

Algorithm 1 depicts the basic process of the collaborative inference optimization framework, which starts with the energy level stabilization process. Once the stabilization condition is met, the energy level transition is triggered, and this process is iteratively repeated until the ground state is reached at the current scale. Subsequently, the scale is adjusted and the aforementioned steps are executed again under the new granularity until the termination condition is reached. This mechanism converts the global optimization problem into a multiscale ground-state convergence problem, allowing for more efficient optimization.

The proposed optimization framework is highly flexible and adaptable to different model and feature compression methods, which can be replaced with other methods as necessary. The core components, such as the PG, AP, DC, and OP, can be easily modified to accommodate different methods. Moreover, the optimization indicators used in the framework can be changed to suit different criteria. For instance, if energy cost is a crucial criterion, the DC can be transformed into an energy calculator.

Algorithm 1: Proposed Collaborative Inference Optimization Framework

Input: k: number of populations; acc_{th}: accuracy threshold; $\iota_{\max}$: maximum iterations; λ: scaling factor.

Output: optimal delay $\tau_{\min}$ and corresponding hyper-parameters $\mathbf{X}_{\min}$.

1 uniformly generate k solutions of hyper-parameters;
2 **for** *each solution* $\boldsymbol{X}_j$ **do**
3 calculate the total delay τ_j related to $\mathbf{X}_j$;
4 **end**
5 initial the optimal delay $\tau_{\min}$ by the minimum τ_j;
6 **while** $\iota < \iota_{\max}$ **do**
7 **while** $\sigma > \sigma_s$ **do**
8 **while** $\Delta\sigma > \sigma_s$ **do**
 // **Energy Level Stabilization**
9 **for** *each solution* $\boldsymbol{X}_j$ **do**
10 generate proper $\mathbf{X}'_j \sim N(\mathbf{X}_j, \sigma_s^2)$;
11 calculate predicted accuracy acc'_j;
12 **if** $acc'_j \leq acc_{th}$ **then**
13 go back to line 10;
14 **end**
15 calculate τ'_j related to $\mathbf{X}'_j$;
16 **if** $\tau'_j \leq \tau_{\min}$ **then**
17 $\tau_{\min} \leftarrow \tau'_j$;
18 $\mathbf{X}_j \leftarrow \mathbf{X}'_j$;
19 **end**
20 **end**
21 calculate standard deviation σ' of all $\mathbf{X}_j$;
22 $\Delta\sigma \leftarrow |\sigma' - \sigma|$;
23 update $\sigma \leftarrow \sigma'$;
24 **end**
 // **Energy Level Transition**
25 replace the worst solution with the mean: $\mathbf{X}^w \leftarrow \mathbf{X}^m$;
26 update standard deviation σ of all $\mathbf{X}_j$;
27 **end**
 // **Scale Adjustment**
28 $\sigma_s \leftarrow \sigma_s/\lambda$;
29 $\iota \leftarrow \iota + 1$;
30 **end**

Such flexibility makes the framework widely applicable and extendable to various applications.

D. Computational Complexity

The computational complexity of the algorithm instance within the proposed framework is noteworthy. Algorithm 1 shows that the quantum-inspired OP is the main body of the framework in this specific algorithm instance.

From Algorithm 1, we can see that the codes of *energy level stabilization* is the main loop code that executes the most. The time complexity of PG in line 10 is $O(k * N_d)$, where N_d represent the dimensions of the problems. When the number of population k is fixed, this time complexity increases linearly with the scale of the problem, that is, the time complexity of this part is $O(N_d)$. The time complexities of AP in line 11 and DC in line 15 are at most $O(N_d)$. Besides, the time complexity of *scale adjustment* does not change with the scale of the problem, that is, the time complexity of this part is $O(1)$. After analyzing the whole algorithm like this, we can see that the time complexities of other parts are at most $O(N_d)$. Based on the above analysis, it can be concluded that the proposed algorithm exhibits linear time complexity as a whole.

TABLE II
DEVICE AND SERVER SPECIFICATIONS

Type	Name	Performance(GFLOPS)
Device	Raspberry Pi 3B+ [34]	0.2
Device	Raspberry Pi 4B [35]	3
Device	NVIDIA Jetson TK1 [36]	326.4
Server	NVIDIA Tesla P100 [37]	10600

VI. SIMULATIONS

In this section, we conduct some simulations to evaluate the performance of the proposed framework and deliver a discussion and analysis of the results.

A. Simulation Scenario

Software simulations are conducted to evaluate the performance of the collaborative inference optimization framework under different hardware computation specifications and wireless communication link conditions.

1) DNN Model: RegNet [32], developed by the Facebook AI team, represents the latest advancement in deep learning architecture. It is a model family that employs a novel network design paradigm to explore design spaces and create innovative, effective architectures. For the proposed collaborative inference optimization framework, we select RegNetX-400MF from the RegNet model family along with AlexNet to assess its effectiveness.

2) Data Set: The CIFAR-10 data set [33] is used to evaluate the performance of the proposed framework. It is a widely used benchmark for evaluating image classification models and comprises 60 000 color images with a resolution of 32 × 32 pixels, distributed across ten different categories with 6000 images per category. The data set is divided into five training batches and one test batch, each containing 10 000 images.

3) Hardware Specifications: Different parameters of devices and the server are shown in Table II.

As presented in Table II, three devices with different performances are considered to evaluate the impact of computation capability on the inference latency. NVIDIA Tesla P100 hardware platform with the capability of 10.6TFLOPS is considered as the server in the simulations.

4) Wireless Link Conditions: The influence of communication conditions on the inference latency is evaluated using four communication links with different average uplink rates from the SPEEDTEST website [38], as shown in Table III.

5) Other Parameters: The simulations were conducted using Python 3.8 and PyTorch 1.12.0, evaluating the AlexNet model (initial accuracy: 87.42%) and the RegNet model (initial

TABLE III
DATA RATES OF WIRELESS COMMUNICATION LINKS

Name	Uplink Rate(Mbps)
3G	1.19
4G	11.22
5G	52.24
WiFi	54.19

TABLE IV
PERFORMANCE OF DIFFERENT REGRESSORS

Regressor	MSE	AR	AIC	BIC
GradientBoostingRegressor	4.87E-05	8.32E-01	-3.90E+03	-3.83E+03
BaggingRegressor	5.69E-05	7.23E-01	-3.75E+03	-3.69E+03
AdaBoostRegressor	4.89E-05	7.01E-01	-3.72E+03	-3.65E+03
RandomForestRegressor	4.81E-05	7.84E-01	-3.81E+03	-3.75E+03
DecisionTreeRegressor	7.56E-05	5.43E-01	-3.66E+03	-3.60E+03
LinearRegression	5.14E-05	7.00E-01	-3.73E+03	-3.67E+03

accuracy: 91.71%). The goal is to achieve optimal latency while maintaining a maximum model accuracy loss of 2%. The other parameters used in the simulations are set as follows: $k = 5$, $\iota_{\max} = 20$, and $\lambda = 1.3$. The compared algorithm parameters are selected according to the recommendations provided in their respective papers. The results are analyzed to provide insight into the framework's performance and effectiveness.

B. Baseline

1) All_Local: The inference task is only performed by the whole model on the device, and wireless data transmission is not required. Thus, the inference delay is only related to the device parameters.

2) All_Server: Since the inference tasks are only performed by the whole model on the server, all input data needs to be transmitted to the server for inference, making the inference delay dependent on the wireless transmission parameters.

3) Neurosurgeon [7]: This scheme involves partitioning the model and deploying its different parts separately on the device and server. The optimal partition point is determined to minimize the delay in collaborative inference.

4) Prune_Partition: The model undergoes layerwise sparsity pruning before being split into two parts. An inference optimization is employed to determine the optimal layerwise sparsity ratios and the partition point for minimizing inference delay.

5) 2_Step_Pruning [39]: This device-edge co-inference scheme employs a two-step pruning approach: 1) pruning the entire pretrained model and 2) pruning the layer preceding the split point.

6) Filter Pruning via Geometric Median (FPGM) [40]: FPGM is a novel filter pruning strategy using the geometric median. It differs from previous norm-based criteria by explicitly considering the interrelations among filters. An auto-selection scheme is incorporated into it to identify the optimal model compression parameters and partition point.

7) BottleNet++ [23]: BottleNet++ splits the model into two parts and uses an encoder–decoder pair to compress features. An auto-selection scheme is incorporated into it to identify the optimal feature compression parameters and partition point, with the goal of minimizing inference delay.

8) Co-Inference via AutoML (CIA) [26]: This is an AutoML framework to achieve communication-computation efficient device-edge co-inference based on deep reinforcement learning (DRL), which determines the sparsity level and compression ratio of each DNN layer.

C. Accuracy Prediction Model

It is crucial to ensure that the generated model parameters meet the desired inference accuracy requirements. Therefore, after generating candidate parameters, the model needs to be trained and tested to verify the accuracy of the generated model. As this process is required for every iteration of the optimization process, it can consume a significant amount of time.

The objective of accuracy prediction is to construct a predictive model capable of estimating the test accuracy of a modified network model, thereby facilitating the identification of optimal parameters that preserve the accuracy of the original network while minimizing computational overhead.

To construct an accuracy prediction model, a data set of 300 test accuracy results is generated by training and evaluating DNN models with different combinations of decision variables. Various regressors from the sklearn Python package [41] are compared on the data sets, and several metrics [42], including the mean squared error (MSE), adjusted R-squared (AR), Akaike information criterion (AIC), and Bayesian information criterion (BIC), are calculated. The results of this comparison are presented in Table IV.

GradientBoostingRegressor is a machine learning technique that employs the gradient boosting approach [43] for regression tasks and is favored for its ease of implementation. After evaluating multiple regression models using various metrics, GradientBoostingRegressor has achieved the highest AR of 8.32E-01, AIC of −3.90E+03, and BIC of −3.83E+03. Although its MSE is comparable to the top-performing regressor, GradientBoostingRegressor has exhibited the best comprehensive performance among the evaluated methods. Hence, it has been selected as the accuracy prediction model for inference accuracy estimation.

It is necessary to rebuild the accuracy prediction model if the decision variables are modified or the noise is introduced to ensure the accuracy of the prediction. Consequently, distinct accuracy prediction models have been generated to account for the different scenarios. The time required for accuracy prediction modeling is dependent on the complexity of the DNN model and the dimensionality of the decision variables. In the modeling process, data collection is the most time-consuming part. Simply adding noise increases the dimension of the decision variables by one, without significantly affecting the overall time cost of accuracy prediction modeling.

An important point to note is that both the modeling of the accuracy prediction model and the optimization problem-solving occur during the offline stage. The final inference process operates independently of these two stages, and the

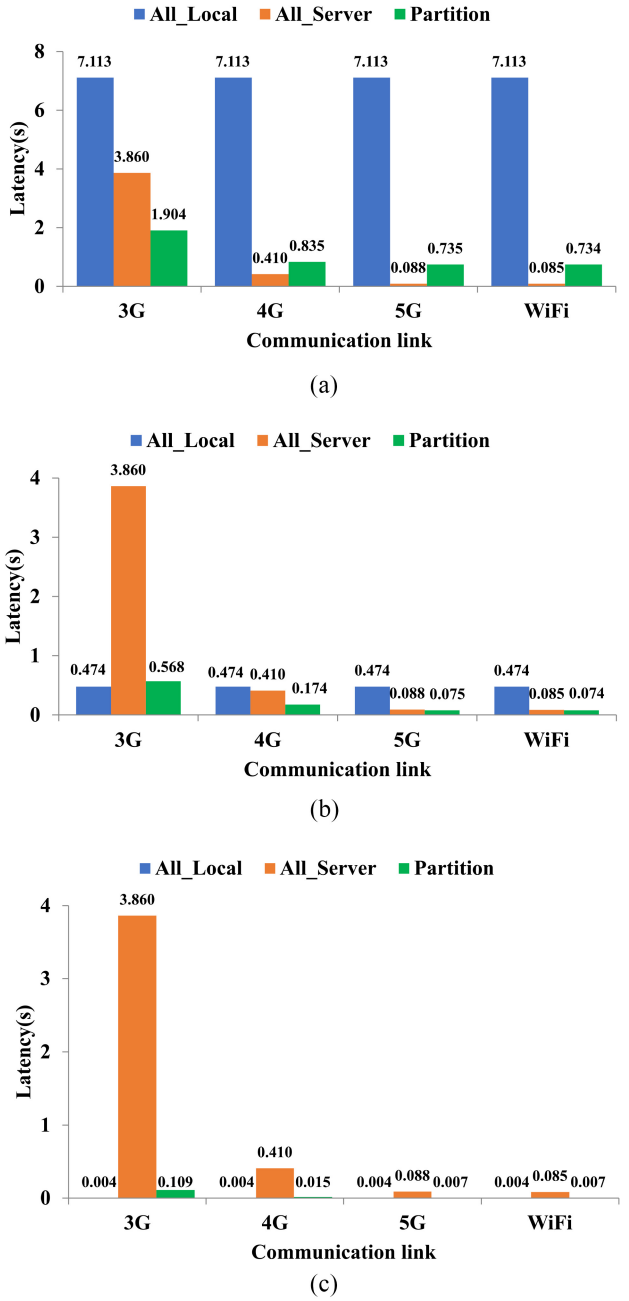


Fig. 6. Inference latency versus wireless communication link. (a) Raspberry Pi 3B+. (b) Raspberry Pi 4B. (c) Jetson TK1.

inference time is not affected by the execution time of them. By using accuracy prediction instead of conventional training and testing methods, the entire framework's running time is significantly accelerated.

D. Discussion

1) Performance of Partitioning: This section investigates the partitioning performance of various hardware specifications and wireless links using different methods. The simulation results are presented in Fig. 6.

Fig. 6(a) illustrates that Partition achieves a maximum inference acceleration of up to 3.7× and a minimum of 2× compared to All_Local and All_Server under slow network speeds (3G). As the data rate increases, Partition shows lower inference latency due to reduced communication latency. Nevertheless, Partition achieves a maximum inference latency reduction of 9.7× and a minimum of 8.5× compared to All_Local under various wireless link conditions, except for 3G.

As shown in Fig. 6(b), All_Server exhibits the slowest performance at low-data rates due to the substantial impact of communication latency on inference latency, while the increased computing performance of Raspberry Pi 4B reduces computation latency. Partition consistently achieves inference acceleration compared to the other methods, with a maximum of 6.4× and a minimum of 1.2×. It effectively balances computation and communication latencies to achieve inference acceleration.

As shown in Fig. 6(c), the inference latency of All_Local exhibits a significant reduction, thus emerging as the fastest method. Partition brings significant inference acceleration by optimizing the partition point to minimize inference latency. It achieves this by reducing inference latency by up to 35× and a minimum of 13× compared to All_Server under different wireless link conditions.

The aforementioned simulation highlights that partitioning can achieve substantial inference acceleration.

2) Communication Rate: This section investigates the impact of communication rates on inference latency for different hardware specifications. We utilize Raspberry Pi 3B+ and Jetson TK1 as devices and Tesla P100 as the server. The communication rate is varied from 0.01 to 100 Mb/s, and the proposed framework is employed to minimize the inference latency. The simulation results for AlexNet and RegNet are presented in Figs. 7 and 8.

The results in Fig. 7(a) demonstrate that as the communication rate increases, the inference latency decreases for all methods except All_Local, which does not require wireless communication for data transmission. At communication rates slower than 0.01 Mb/s, the scheme incorporating feature compression demonstrates superior acceleration performance. Our method outperforms all other methods in terms of acceleration. The results indicate that when attempting to accelerate inference through the reduction of transmission data size, relying solely on partitioning and model compression may not be sufficient, particularly when the communication rate is low.

As the communication rate increases, All_Server outperforms other algorithms at very high-communication rates. In other scenarios, our method achieves substantial inference acceleration in most scenarios, making it the most effective approach. Notably, it achieves a substantial acceleration effect with improvements of up to 2.2× (minimum of 1.1×) compared to suboptimal methods and up to 198.0× (minimum of 14.5×) compared to the worst-performing method.

As shown in Fig. 7(b), all methods except for All_Local which does not require wireless communication, exhibit decreasing inference latencies with increasing communication rates. The superior computing performance of Jetson TK1 significantly reduces computation latency. The proposed method demonstrates superior performance and can greatly expedite inference, particularly at low-data transmission speeds.

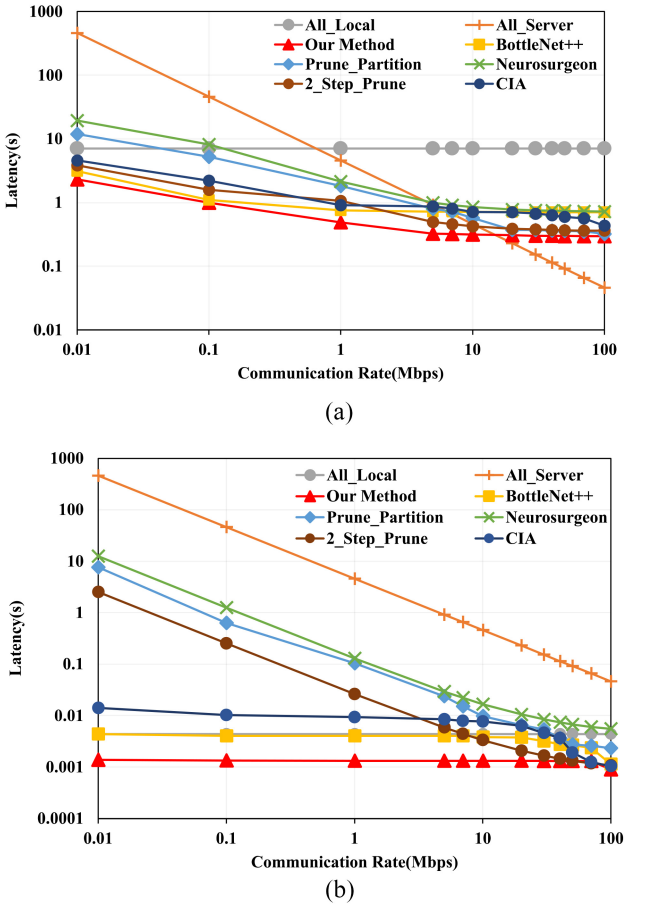


Fig. 7. Inference latency versus communication rate (AlexNet). (a) RaspberryPi 3B+. (b) Jetson TK1.

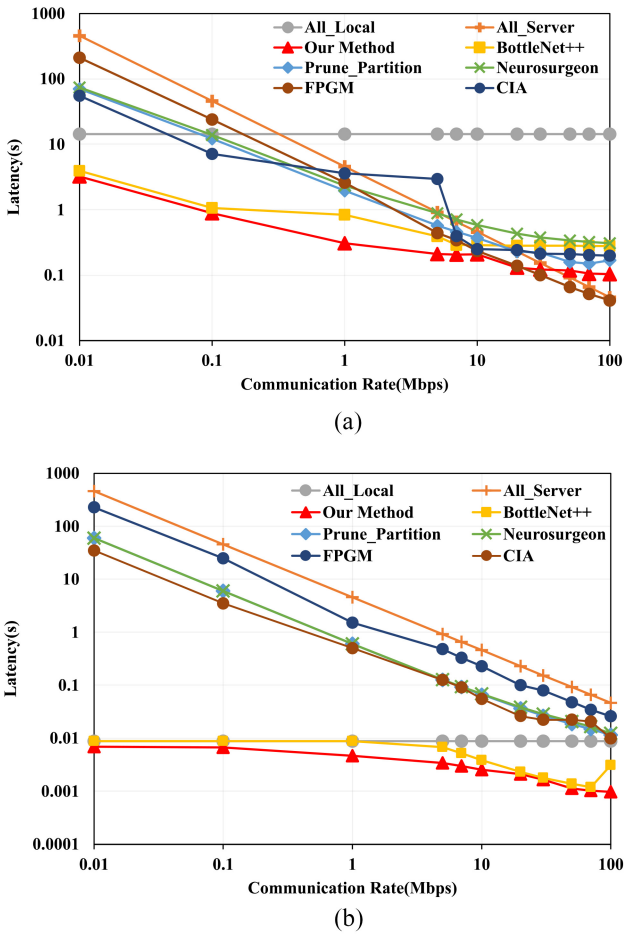


Fig. 8. Inference latency versus communication rate (RegNet). (a) RaspberryPi 3B+. (b) Jetson TK1.

Compared to suboptimal methods, our approach achieves inference acceleration ranging from 1.1× to 3.1×. In comparison to the worst-performing method, the acceleration ranges from 51.2× to over 10^6×, demonstrating significant performance improvement.

As shown in Fig. 8(a), the overall trend of all curves is consistent with the simulation results of AlexNet. When the device has limited computing power, the impact of communication latency on inference is reduced as the communication rate increases. Due to the significant discrepancy in computing performance between Raspberry Pi 3B+ and Tesla P100, All_Server outperforms other algorithms at high-communication rates. Furthermore, FPGM incorporates an efficient model compression scheme that significantly reduces computation latency, thus accelerating inference in scenarios where the impact of communication latency is slight.

In other scenarios, our method achieves substantial inference acceleration. It demonstrates improvements ranging from 1.1× to 2.7× compared to suboptimal methods, and from 46.4× to 141.0× compared to the worst-performing method.

As shown in Fig. 8(b), our method consistently outperforms in all scenarios, delivering superior performance. It achieves acceleration with improvements ranging from 1.1× to 3.2× compared to suboptimal methods, and from 47.4× to more than 10^5× compared to the worst-performing method.

These results highlight the effectiveness of our approach across various DNN models and emphasize the importance of carefully selecting the appropriate optimization strategy for achieving optimal acceleration performance.

3) Communication-Computation Trade-Off: In this section, the communication overhead and local computation of compared methods at different partition points are investigated. The device parameters remain consistent with Raspberry Pi 3B+, while the server parameters are consistent with Tesla P100. The communication rate is set to 11.22 Mb/s (4G). The results for AlexNet and RegNet are presented in Figs. 9 and 10.

As depicted in Figs. 9 and 10, The methods utilizing model compression substantially decrease local computation, causing a greater concentration of data points toward the left compared to other methods, indicating reduced computational load. The data points corresponding to the methods employing feature compression are predominantly situated at the bottom of the figure, indicating lower communication overhead compared to the other methods.

Our method shows a better tradeoff between communication overhead and local computation according to the results for AlexNet and RegNet, as the majority of its data points are located to the left and bottom of the other methods. This suggests that our method achieves more balanced compression

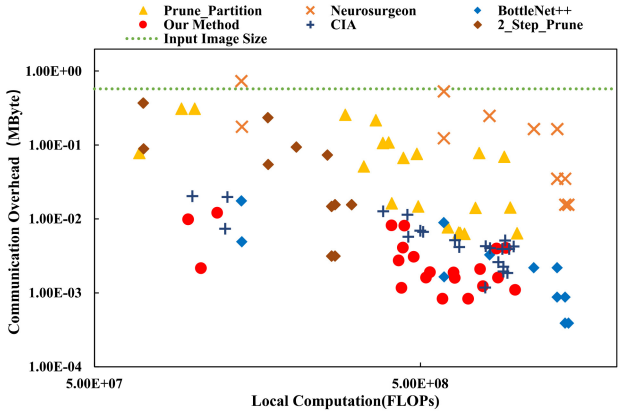


Fig. 9. Communication overhead versus local computation (AlexNet).

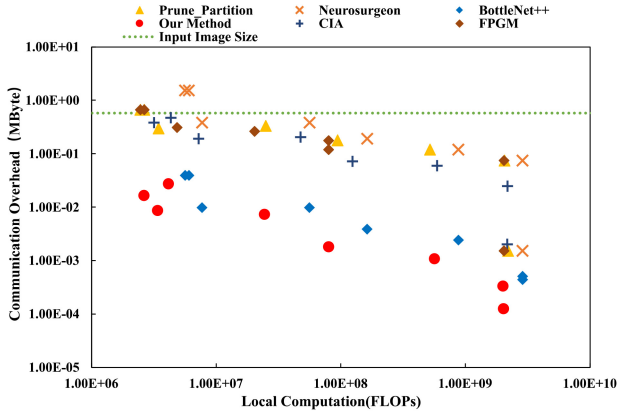


Fig. 10. Communication overhead versus local computation (RegNet).

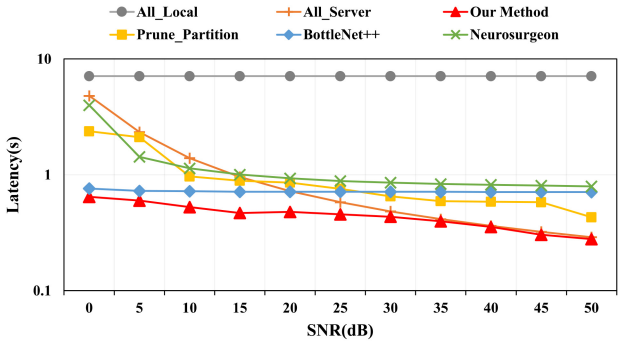


Fig. 11. Inference latency versus SNR.

in both communication and local computation than the other methods.

4) Effect of Noise: To investigate the impact of wireless channel noise on communication performance and inference latency, we consider the additive white Gaussian noise (AWGN) channel [23] and use SNR to indicate the channel condition. Specifically, we maintain the device parameters consistent with Raspberry Pi 3B+, while the server parameters are set to those of Tesla P100. Our method is compared with selected methods across a wide range of SNR levels spanning from 0 to 50 dB for the inference task involving AlexNet. The results of the evaluation are presented in Fig. 11.

Fig. 11 demonstrates that the inference acceleration methods influenced by communication latency exhibit improved performance as the SNR increases. However, All_Local remains unaffected by SNR since it does not require wireless links for data transmission, as mentioned earlier. The increase in SNR leads to a rise in the data transmission rate, resulting in a substantial reduction of latency for the methods which utilize model compression. Methods that utilize feature compression to reduce output data size for reducing communication latency exhibit relative stability across different channel conditions. However, our method outperforms all other methods in terms of maintaining the minimum inference latency even under extremely low-SNR conditions.

Specifically, our method achieves the inference acceleration by up to 1.5× and at least 1.02× compared to the suboptimal method, and by up to 25.5× and at least 11.0× compared to the worst method, respectively.

VII. Conclusion

This article proposes a novel approach to address the inference acceleration problem, specifically by introducing the concept of collaborative device-edge inference with three primary steps involved: 1) model compression; 2) DNN partitioning; and 3) feature compression. The collaborative inference latency optimization problem is formulated as a mixed integer programming problem, which involves defining numerous hyperparameters to identify the optimal decision variable set. The challenge of optimizing collaborative inference latency is tackled by proposing a framework comprising four fundamental modules: 1) PG; 2) AP; 3) DC; and 4) OP. An inference accuracy prediction model using multiple regression is built to evaluate the candidate hyperparameter set's accuracy. This study evaluates the proposed collaborative inference optimization framework through software simulations, using the inference tasks of AlexNet and RegNet on CIFAR-10 as an instance, under different hardware computation specifications and wireless communication link conditions. The simulation results demonstrate the effectiveness and superior performance of the proposed framework in achieving significant inference acceleration compared to alternative methods in various scenarios.

The proposed framework corresponds to a family of algorithms when choosing different technology architecture and it offers a flexible architecture for optimizing collaborative inference latency. For instance, analyzing the sparsity pattern of a particular DNN model can improve pruning performance, while quantization and other model compression techniques can reduce computation latency. More advanced encoding and decoding techniques can also be implemented to reduce transmission latency. By adjusting the hyperparameters and corresponding modules in the framework, it is expected that a more effective optimization strategy can be easily achieved.

This article suggests several promising directions for future research. One is to further optimize model compression and transmission encoding by neural architecture search (NAS) technology. Another is to implement the proposed framework on real-world hardware platforms with dynamic collaborative inference environments, utilizing reinforcement learning and other intelligent technologies.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[4] A. A. Nugraha, A. Liutkus, and E. Vincent, "Multichannel audio source separation with deep neural networks," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 24, no. 9, pp. 1652–1664, Sep. 2016.

[5] H.-C. Shin et al., "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1285–1298, May 2016.

[6] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[7] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017.

[8] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020.

[9] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018.

[10] X. Chen, M. Li, H. Zhong, Y. Ma, and C.-H. Hsu, "DNNOff: Offloading DNN-based intelligent IoT applications in mobile edge computing," *IEEE Trans. Ind. Informat.*, vol. 18, no. 4, pp. 2820–2829, Apr. 2022.

[11] G. Liu et al., "An adaptive DNN inference acceleration framework with end–edge–cloud collaborative computing," *Future Gener. Comput. Syst.*, vol. 140, pp. 422–435, Mar. 2023.

[12] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," 2019, *arXiv:1912.12115*.

[13] N. Shlezinger and I. V. Bajić, "Collaborative inference for AI-empowered IoT devices," *IEEE Internet Things Mag.*, vol. 5, no. 4, pp. 92–98, Dec. 2022.

[14] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief, "Communication-efficient edge AI: Algorithms and systems," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 4, pp. 2167–2191, 4th Quart., 2020.

[15] M. D. Emmerson and R. I. Damper, "Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application," *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 788–793, Sep. 1993.

[16] J. Shao and J. Zhang, "Communication-computation trade-off in resource-constrained edge inference," *IEEE Commun. Mag.*, vol. 58, no. 12, pp. 20–26, Dec. 2020.

[17] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2016, *arXiv:1608.08710*.

[18] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence," *Information*, vol. 11, no. 4, p. 193, 2020.

[19] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin, "Layer-adaptive sparsity for the magnitude-based pruning," 2020, *arXiv:2010.07611*.

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015, *arXiv:1510.00149*.

[21] 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 *Proc. IEEE 24th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, 2018, pp. 671–678.

[22] E. Bourtsoulatze, D. B. Kurka, and D. Gündüz, "Deep joint source-channel coding for wireless image transmission," *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 3, pp. 567–579, Sep. 2019.

[23] J. Shao and J. Zhang, "Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, 2020, pp. 1–6.

[24] A. E. Eshratifar, A. Esmaili, and M. Pedram, "BottleNet: A deep learning architecture for intelligent mobile cloud computing services," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, 2019, pp. 1–6.

[25] Y. Matsubara and M. Levorato, "Neural compression and filtering for edge-assisted real-time object detection in challenged networks," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, 2021, pp. 2272–2279.

[26] X. Zhang, J. Shao, Y. Mao, and J. Zhang, "Communication-computation efficient device-edge co-inference via AutoML," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2021, pp. 1–6.

[27] Y. K. Tun, Y. M. Park, N. H. Tran, W. Saad, S. R. Pandey, and C. S. Hong, "Energy-efficient resource management in UAV-assisted mobile edge computing," *IEEE Commun. Lett.*, vol. 25, no. 1, pp. 249–253, Jan. 2021.

[28] P. Wang, X. Ye, B. Li, and K. Cheng, "Multi-scale quantum harmonic oscillator algorithm for global numerical optimization," *Appl. Soft Comput.*, vol. 69, pp. 655–670, Aug. 2018.

[29] A. B. Finnila, M. A. Gomez, C. Sebenik, C. Stenson, and J. D. Doll, "Quantum annealing: A new method for minimizing multidimensional functions," *Chem. Phys. Lett.*, vol. 219, nos. 5–6, pp. 343–348, 1994.

[30] J. Sun, W. Fang, X. Wu, V. Palade, and W. Xu, "Quantum-behaved particle swarm optimization: Analysis of individual particle behavior and parameter selection," *Evol. Comput.*, vol. 20, no. 3, pp. 349–393, Sep. 2012.

[31] J. Brooke, D. Bitko, T. F. Rosenbaum, and G. Aeppli, "Quantum annealing of a disordered magnet," *Science*, vol. 284, no. 5415, pp. 779–781, 1999.

[32] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10428–10436.

[33] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[34] R. PI. "Raspberry Pi 3 model B+." 2022. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

[35] R. PI. "Raspberry Pi 4." 2022. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[36] NVIDIA. "NVIDIA Jetson TK1 module." 2022. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/industries/automotive/documents/jetson-tk1-module-datasheet.pdf

[37] NVIDIA. "Tesla P100data center accelerator." 2022. [Online]. Available: https://www.nvidia.com/en-us/data-center/tesla-p100/

[38] Ookla. "Speedtest by Ookla - the global broadband speed test." 2022. [Online]. Available: https://www.speedtest.net/global-index/united-states-mobile-data/2020/q2

[39] 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 *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 1–6.

[40] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4340–4349.

[41] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, 2011.

[42] C. A. Escobar and R. Morales-Menendez, "Process-monitoring-for-quality—A model selection criterion for l_1-regularized logistic regression," *Procedia Manuf.*, vol. 34, pp. 832–839, Jan. 2019.

[43] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artif. Intell. Rev.*, vol. 54, no. 3, pp. 1937–1967, 2021.

Lei Mu received the Ph.D. degree from Beijing Institute of Technology, Beijing, China, in 2011.

From 2022 to 2023, he was a Visiting Research Fellow with the Department of Engineering, King's College London, London, U.K. He is currently a Lecturer with the School of Computer Science and Engineering, Southwest Minzu University, Chengdu, China. His current research interests include edge intelligence, distributed learning, and swarm intelligence.

Zhonghui Li received the M.S. degree from Southwest Minzu University, Chengdu, China, in 2023.

He is currently working as a Teacher with Hubei Enshi College, Enshi, China. His main research interests include deep neural networks, mobile edge computing, and computing offloading.

Wei Xiao is currently pursuing the M.S. degree with Eötvös Loránd University, Budapest, Hungary.

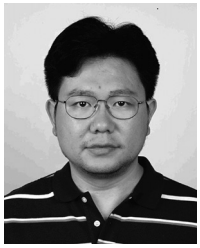
His major research interests include artificial intelligence and Internet of Things.

Ruilin Zhang is currently pursuing the M.S. degree with the University of Auckland, Auckland, New Zealand.

Her research interests mainly include quantitative research methods in information systems and Internet of Things in industrial digitization.

Peng Wang received the B.Eng. and M.S. degrees from Sichuan University, Chengdu, China, in 1998 and 2001, respectively, and the Ph.D. degree from the Institute of Computer Application, Chinese Academy of Sciences, Chengdu, in 2004.

He is a Full Professor with Southwest Minzu University, Chengdu, and the Ph.D. Advisor with the Institute of Computer Application, Chinese Academy of Sciences. His research interests include computational intelligence, high-performance computing, and quantum-inspired algorithms.



Tao Liu received the B.S. degree in computer science from Nanjing University of Posts and Telecommunications, Nanjing, China, in 2000, the M.S. degree in computer science from the University of Electronic and Technology of China, Chengdu, China, in 2004, and the Ph.D. degree in computer science from Sichuan University, Chengdu, in 2012.

He is currently a Professor with the School of Computer Science and Engineering, Southwest Minzu University, Chengdu. He was a Visiting Scholar with Columbia University, New York, NY, USA, from August 2015 to August 2016. His research interests include ad hoc networks, wireless sensor networks, and Internet of Things.

Geyong Min (Member, IEEE) received the B.Sc. degree in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1995, and the Ph.D. degree in computing science from the University of Glasgow, Glasgow, U.K., in 2003.

He is a Professor of High-Performance Computing and Networking with the Department of Computer Science, University of Exeter, Exeter, U.K. His research interests include computer networks, wireless communications, parallel and distributed computing, ubiquitous computing, multimedia systems, and modeling and performance engineering.

Keqin Li (Fellow, IEEE) received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 1985, and the Ph.D. degree in computer science from the University of Houston, Houston, TX, USA, in 1990.

He is currently a SUNY Distinguished Professor with the State University of New York, New Paltz, NY, USA, and a National Distinguished Professor with Hunan University, Changsha, China. He has authored or coauthored more than 940 journal articles, book chapters, and refereed conference papers. He holds nearly 70 patents announced or authorized by the Chinese National Intellectual Property Administration.

Prof. Li received several best paper awards from international conferences, including PDPTA-1996, NAECON-1997, IPDPS-2000, ISPA-2016, NPC-2019, ISPA-2019, and CPSCom-2022. He is among the world's top five most influential scientists in parallel and distributed computing in terms of both single-year impact and career-long impact based on a composite indicator of Scopus Citation Database. He was a 2017 recipient of the Albert Nelson Marquis Lifetime Achievement Award for being listed in Marquis Who's *Who in Science and Engineering*, Who's *Who in America*, Who's *Who in the World*, and Who's *Who in American Education* for more than 20 consecutive years. He received the Distinguished Alumnus Award of the Computer Science Department at the University of Houston in 2018. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022, and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He is a member of SUNY Distinguished Academy. He is an AAAS Fellow and an AAIA Fellow. He is a member of Academia Europaea (Academician of the Academy of Europe).