DeViT: Decomposing Vision Transformers for Collaborative Inference in Edge Devices

Guanyu Xu , Zhiwei Hao , Yong Luo , *Member, IEEE*, Han Hu , *Member, IEEE*, Jianping An , *Member, IEEE*, and Shiwen Mao , *Fellow, IEEE*

Abstract—Recent years have witnessed the great success of vision transformer (ViT), which has achieved state-of-the-art performance on multiple computer vision benchmarks. However, ViT models suffer from vast amounts of parameters and high computation cost, leading to difficult deployment on resource-constrained edge devices. Existing solutions mostly compress ViT models to a compact model but still cannot achieve real-time inference. To tackle this issue, we propose to explore the divisibility of transformer structure, and decompose the large ViT into multiple small models for collaborative inference at edge devices. Our objective is to achieve fast and energy-efficient collaborative inference while maintaining comparable accuracy compared with large ViTs. To this end, we first propose a collaborative inference framework termed DeViT to facilitate edge deployment by decomposing large ViTs. Subsequently, we design a decomposition-and-ensemble algorithm based on knowledge distillation, termed DEKD, to fuse multiple small decomposed models while dramatically reducing communication overheads, and handle heterogeneous models by developing a feature matching module to promote the imitations of decomposed models from the large ViT. Extensive experiments for three representative ViT backbones on four widely-used datasets demonstrate our method achieves efficient collaborative inference for ViTs and outperforms existing lightweight ViTs, striking a good trade-off between efficiency and accuracy. For example, our DeViTs improves end-to-end latency by 2.89× with only 1.65% accuracy sacrifice using CIFAR-100 compared to the large ViT, ViT-L/16, on the GPU server. DeDeiTs surpasses the recent efficient ViT, MobileViT-S, by 3.54% in accuracy on ImageNet-1 K, while running 1.72× faster and requiring 55.28% lower energy consumption on the edge device.

Index Terms—Collaborative inference, edge computing, model decomposition, vision transformer.

I. INTRODUCTION

TRANSFORMERS [1] have been widely used in natural language processing (NLP) [2], achieved exceptional performance in audio tasks [3], and demonstrated their great potentials in computer vision [4], [5]. With the rapid development of the artificial intelligence of things (AIoT), deploying transformer-based models on resource-limited edge devices has becoming an attractive idea. However, the inference cost of transformer is very high, leading to unacceptable latency or energy consumption, especially for resource-limited edge devices [6]. For example, the ViT-L/16 model [5], a typical vision transformer, requires 1.12 GB storage and 190.7 Giga Floating Point Operations (GFLOPs) for inference, while a Raspberry Pi-4B device only has 4 GB RAM and 13.5 Giga Floating Point Operations Per Second (GFLOPS) [7], [8]. In order to achieve effective inference, a pre-trained model is usually divided into multiple parts, and each part is deployed on an edge device or a server. These devices conduct inference collaboratively. This is often called collaborative inference, which avoids transmitting large data to the server, and can substantially improve the quality of experience in the AIoT [9].

In collaborative inference, some methods partition DNNs by or across layers [9], [10]. The intermediate features should be transmitted to devices or servers multiple times, resulting in high latency and energy costs. The data transmission can also be easily affected by the network dynamics. Furthermore, some complex surgery strategies are usually required to find a proper point to decouple the original model [11]. Therefore, achieving fast and effective collaborative inference on edge devices is challenging.

Vision transformers (ViTs) cannot be directly deployed on edge devices because of the large model size and high computation cost. To tackle this problem, some existing works utilize conventional model compression methods to assist the deployment of large ViTs and achieve fast inference. These methods need to elaborately design [12] or search [13] for the compact structure, and cannot balance between accuracy and the constrained resource requirements of edge devices. This is because these methods focus on reducing model parameters and FLOPs, which yet are not the right criterion for the ultimate goals of latency and energy. Moreover, almost no on-device experiments have been conducted so far to measure the actual inference latency and energy consumption.

To remedy these limitations, we propose a collaborative inference framework for general ViTs in edge devices, termed DeViT,

by decomposing the large ViT into multiple small models. These decomposed models can be readily deployed on edge devices for conducting inference simultaneously, which significantly reduces inference latency and energy consumption with slight accuracy drop. To the best of our knowledge, this is the first work to achieve collaborative inference for ViTs in edge devices by leveraging the divisibility of the transformer structure. Subsequently, we propose an efficient decomposition-and-ensemble algorithm based on knowledge distillation, termed DEKD, to counterbalance the accuracy loss induced by decomposition and facilitate communication overheads among edge devices. Each device equipped with a decomposed model only needs to execute a singular transfer of target features for aggregation, resulting in a reduction of transmission costs alongside a satisfactory ensemble performance. To ensure acceptable performance, we also design a novel feature matching module to mitigate information in DEKD algorithm, which promotes the imitations of decomposed models from the large transformer by matching the dimension of intermediate features between them.

We conduct comprehensive experiments on both GPU servers and edge devices to verify the effectiveness of our proposed framework. In the experiments, we first decompose a large ViT according to the number of edge devices and their hardware configuration. Then we utilize DEKD to retain the most important parts in the small models by shrinking and finally fuse these small models by employing our feature aggregation module. The results show that the proposed DeViT framework can achieve real-time inference and almost no accuracy drop on edge devices for large ViTs. Moreover, we conduct massive experiments using three ViT backbones and four computer vision datasets to verify the effectiveness of our approach. For instance, on the GPU server, our DeViTs improves end-to-end latency by 2.89× with only 1.65% accuracy sacrifice compared to the large ViT, ViT-L/16 [5], using ImageNet. On the edge device, DeDeiTs surpasses the recent efficient ViT, MobileViT-S [12], by 3.54% in accuracy, while running 1.72× faster and requiring 55.28% lower energy consumption.

Our main contributions in this paper are summarized as follows:

- We propose a collaborative inference framework for general ViTs in edge devices, termed DeViT, by decomposing the large ViT into multiple small models.
- We develop an efficient decomposition-and-ensemble algorithm based on knowledge distillation, termed DEKD, to fuse multiple decomposed models and minimize the accuracy loss caused by decomposition, while dramatically reducing communication overheads among edge devices.
- We design a novel feature matching module that facilitates the learning of decomposed models from the large ViT and minimizes information loss.
- We evaluate our DeViT framework for three representative ViT backbones on both commodity GPU servers and real edge devices using four widely-used datasets. The results demonstrate that our method can reduce inference latency by 1.90× and reduce energy consumption by 26.11% on average with only 2 % accuracy sacrifice.
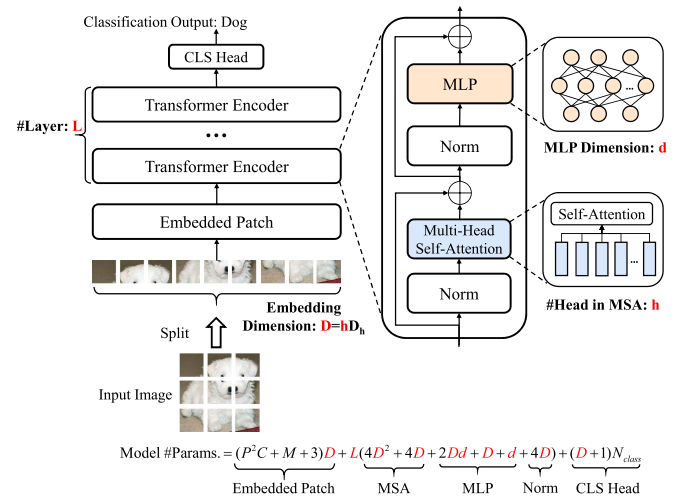
Fig. 1.   Structure of a ViT [5]. ViT is composed of a patch embedding layer, several transformer encoders, and a classification head. The model size mainly depends on the number of encoders, the embedding dimension, the number of attention heads, and the MLP dimension.

The remainder of this paper is organized as follows. We elaborate the challenges for edge deployment of ViTs in Section II. The proposed collaborative deployment framework of ViTs on multiple edge devices is presented in Section III. The proposed DEKD algorithm is described in Section IV, and then Section V provides our experiments and analysis. Finally, we discuss the related works in Section VI and conclude our paper in Section VII.

II. CHALLENGES OF EDGE DEPLOYMENT FOR VITS

In this section, we analyze the number of parameters for ViTs, and discuss the challenges of edge deployment.

A. Parameter Analysis of ViTs

The ViT model consists of a patch embedding layer, multiple stacked transformer encoders, and a classification head, as shown in Fig. 1. The number of parameters depends on the embedding dimension D, the number of transformer encoder L, MLP hidden dimension d, and the number of heads h. The parameters of the embedded patch are $(P^2C + M + 3)D$, where P, C and M is the size of patches, the number of channel and the number of patches, respectively. In a transformer encoder, the parameters of MSA and MLP are $4D^2 + 4D$ and $2Dd + D + d$, respectively. The number of parameters of the two LN modules are $4D$. Given the classification task, we suppose the number of classes is N_{class}. To sum up, the total number of parameters for the entire model is

$$\begin{aligned}
\#\text{Params.} = {} & (P^2C + M + 3)D \\
& + L(4D^2 + 4D + 2Dd + D + d + 4D) \\
& + (D + 1)N_{class},
\end{aligned}$$

(1)

TABLE I
COMPARISON BETWEEN RESOURCES ON TYPICAL EDGE DEVICES AND DEPLOYMENT REQUIREMENTS FOR REPRESENTATIVE ViTs

Resources on Typical Edge Devices				Required Computation Consumption for Deploying Representative ViTs			
Devices	Memory	Computation Performance	TDP	Models	#Parameter	Storage	FLOPs
Raspberry Pi-4B [8]	4 GB	13.5 GFLOPS	7.3 W	DeiT-B [14]	86 M	333 MB	55.4 G
NVIDIA Jetson Nano [7]	4 GB	472 GFLOPS	10 W	Swin-B [4]	88 M	348 MB	47.0 G
NVIDIA Jetson TX2 [7]	8 GB	1.33 TFLOPS	15 W	Swin-L [4]	197 M	763 MB	103.9 G
NVIDIA Jetson Xavier NX [15]	16 GB	21 TOPS	20 W	ViT-L/16 [5]	304 M	1.12 GB	190.7 G

where $D = hD_h$. The model size mainly depends on the number of network layer L, the embedding dimension D, the number of attention heads h, and the MLP dimension d. Taking the ViT-L/16 to perform a 1000-class classification task as an example, the size of patches, the number of patches, the number of network layer, the embedding dimension, the number of attention heads and the MLP dimension are 16, 196, 24, 1024, 16, and 4096, respectively. Thus, the number of ViT-L/16 parameters is 304 M. If the parameters of ViT-L/16 are stored as the 32-bit floating point, it requires a storage space of 1.12 GB.

B. Challenges to Edge Deployment

Although ViTs have achieved the state-of-the-art performance in most vision tasks, it is difficult to deploy them on resource-constrained devices. We compare the resources of typical edge devices with the deployment requirements for representative ViTs in Table I, where we observe three major challenges:

- *Limited memory and storage capacity:* Different from GPU servers, edge devices usually have low memory and storage capacity. For example, ViT-L/16 needs at least 2.60 GB memory and 1.12 GB storage space for inference, while the Raspberry Pi-4B device has only 4 GB memory.
- *Huge computation cost:* The computation cost of ViTs is massive due to their complicated stacked computational structure. For instance, inference with ViT-L/16 requires 190.7 GFLOPs, but the computation capacity of Raspberry Pi-4B is only 13.5 GFLOPS.
- *High inference latency and energy cost:* The inference of ViTs brings about high latency and energy cost. In order to classify an image with 224 224 resolution, the ViT-L/16 model takes 56.79 milliseconds and 0.58 J on an NVIDIA Jetson TX2 device.

In consideration of these challenges, we aim to design a collaborative edge deployment framework for ViTs with the following design goals:

- *Generality:* The framework can be applied to different variants of ViTs.
- *Lightweight:* The framework can accommodate different hardware platforms by reducing the storage and computation cost.
- *Rapidity:* The framework can meet the latency requirement of IoT applications with a fast inference speed.
- *Reliability:* The framework can achieve a performance comparable to that of powerful giant ViTs.
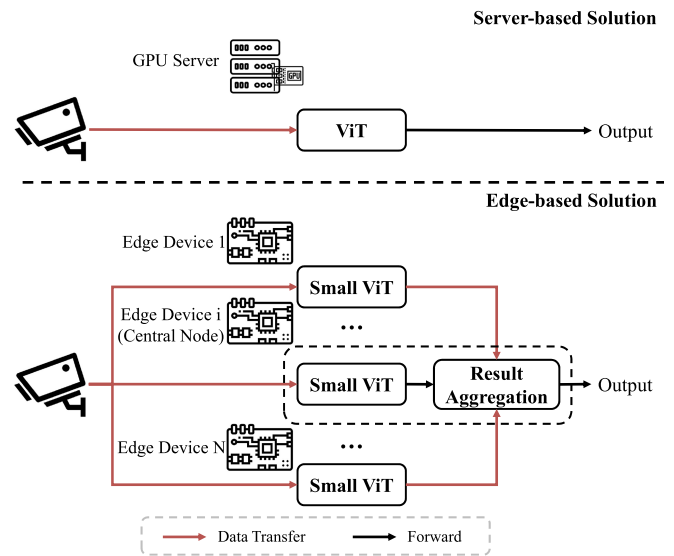
Fig. 2. Comparison of server-based solution (top) and edge-based solution (bottom) for ViT deployment. For sever-based solution, data need to be uploaded to the cloud. The GPU sever executes ViT inference. For edge-based solution, it consists of multiple edge devices with deployed small ViT models. Nearby edge devices execute inference after receiving data. Then, the central device collects the intermediate results from all other devices and aggregates them to produce the inference output.

III. COLLABORATIVE DEPLOYMENT OF ViTs ON MULTIPLE EDGE DEVICES

In this section, we design a collaborative deployment framework of ViT on multiple edge devices. We first introduce the traditional deployment solution on the server. Then we present the workflow of our framework and depict the details of each module in our framework.

A. System Structure and Workflow

In light of the high inference latency and energy consumption, ViTs are usually deployed on commodity GPU servers, as shown in Fig. 2 (top). The workflow of such server-based solution is: initially a giant *ViT* model is deployed on a GPU server in a cloud data center; then the input data (e.g., images, videos) from remote users are uploaded to the cloud; the GPU server runs the ViT model for inference and transmits the results back to the users. This method is convenient to deploy but leads to high transmission cost and latency from the remote users to the cloud.

To tackle this issue, we propose the idea that data are directly transmitted to nearby edge devices to reduce the communication

cost, where these devices are collaboratively deployed for ViTs to improve inference performance. This is achieved by the designed collaborative deployment framework of ViTs on multiple edge devices as shown in Fig. 2 (bottom). It contains multiple edge devices as well as a central node. All the devices collaboratively conduct inference and can communicate with each other via a wireless network such as Wi-Fi, Zigbee, and Bluetooth. The *small ViTs* are deployed on edge devices. When receiving an inference task, the collected data are first transmitted to all edge devices. These devices conduct inference with the *small ViTs* in parallel. Intermediate results of the *small ViTs* from all other edge devices are transmitted to the central edge device, where the *result aggregation* module aggregates the intermediate results to produce the inference output.

The aforementioned framework mainly depends on two modules: small ViT and result aggregation. These modules are required to achieve the following goals:

- *Goal 1:* Small ViTs can be deployed on edge devices and achieve low inference latency.
- *Goal 2:* The results of small ViTs can be aggregated to achieve a performance comparable to that of the powerful giant ViT.

B. Decomposition and Aggregation

We illustrate how to decompose a giant ViT into small models and then aggregate intermediate results from small models to minimize accuracy loss.

Model Decomposition: Considering the Goal 1, we propose to remove the redundant parts in the ViT to get small ViTs. Clark et al. [16] observed that particular heads correspond to specific semantic relations. Michel et al. [17] found that performance are significantly affected by only a small quantity of heads. Khakzar et al. [18] also pointed out that plenty of neurons are unnecessary for specific tasks. Hence, we can discard the redundant structures in the ViT according to the target task and hardware requirement, and decompose the giant ViT into several smaller models. Every decomposed small model is under the hardware and latency constraints. The redundancy of ViT are mainly related to the embedding dimension, the number of attention heads, the MLP dimension, and the number of network layers. We focus on these factors to conduct decomposition.

As shown in Fig. 3, the process of decomposition is as follows:

- *Encoder reduction:* The number of transformer encoders L is decreased to L_s.
- *Head reduction:* Some redundant heads in the MSA module are removed and the reduced number of heads is h_s.
- *Embedding reduction:* The embedding dimension is changed to $D_s = 64h_s$, in order to match the feature dimension in the inference.
- *Neuron reduction:* Some redundant intermediate neurons in the MLP block are discarded and the dimension of MLP is changed from d to d_s.

The reduction criterion will be discussed in Section IV-C. In order to avoid the case that some devices have completed inference but others have not, we only focus on deployment of
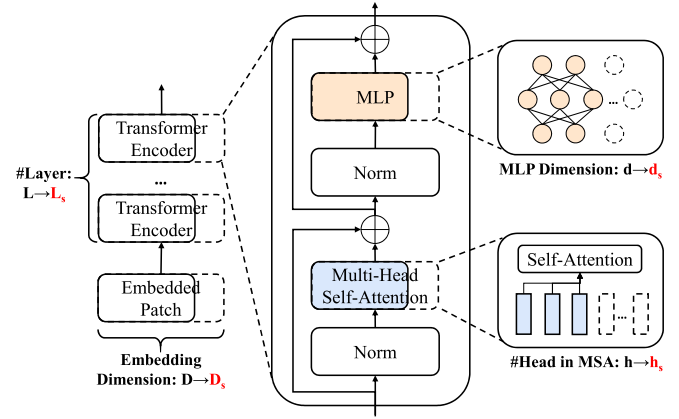

Fig. 3. Process of model decomposition. We focus on embedding dimension, the number of attention heads, the MLP dimension, and the number of network layer when conducting decomposition. The number of ViT layers is decreased from L to L_s. Then redundant heads in MSA and redundant neurons in MLP are removed. In order to match feature dimension, the dimension of embeddings is also reduced to D_s. The dimension of MLP is changed from d to d_s.

homogeneous devices in this work and make structures of all decomposed models the same.

Feature Aggregation: Regarding the Goal 2, we propose to utilize ensemble methods to aggregate results. Most ensemble methods [19] only fuse predictions of different sub-models or select a reliable prediction. However, the final results of these methods mainly depend on the performance of sub-models. The ensemble model cannot achieve satisfactory performance due to the limited performance of sub-models. Hence, we propose to break the performance restrictions of sub-models by aggregating the features of these models, and design an efficient feature aggregation module $FA(\cdot)$ to fuse intermediate features. In this module, the intermediate features from all edge devices are first concatenated, as:

$$\mathbf{X} = \text{Concat}\left(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N\right), \quad (2)$$

where $\mathbf{X}_i, i \in [1, N]$ are the intermediate features of small models. The number of small models is N. The concatenated feature $\mathbf{X} \in \mathbb{R}^{ND}$ is passed to an MLP block consisting of two fully-connected layers. The linear transformation $\mathbf{W}_1 \in \mathbb{R}^{ND \times D_t}$ of the first fully-connected layer allows the interaction of different features. The transformed features can restore the initial dimension by utilizing the transformation $\mathbf{W}_2 \in \mathbb{R}^{D_t \times ND}$ of the second fully-connected layer. The feature aggregation module $FA(\cdot)$ can be formulated as:

$$\begin{aligned} \mathbf{X}_{fused} &= FA(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N) \\ &= (\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2. \end{aligned} \quad (3)$$

We remove the original activation function in the MLP block. This can help decrease the inference latency dramatically and incur almost no accuracy loss. The results in Section V-D show that our aggregation module can guarantee a satisfactory performance and only increase extremely low inference latency compared to other typical aggregation approaches.
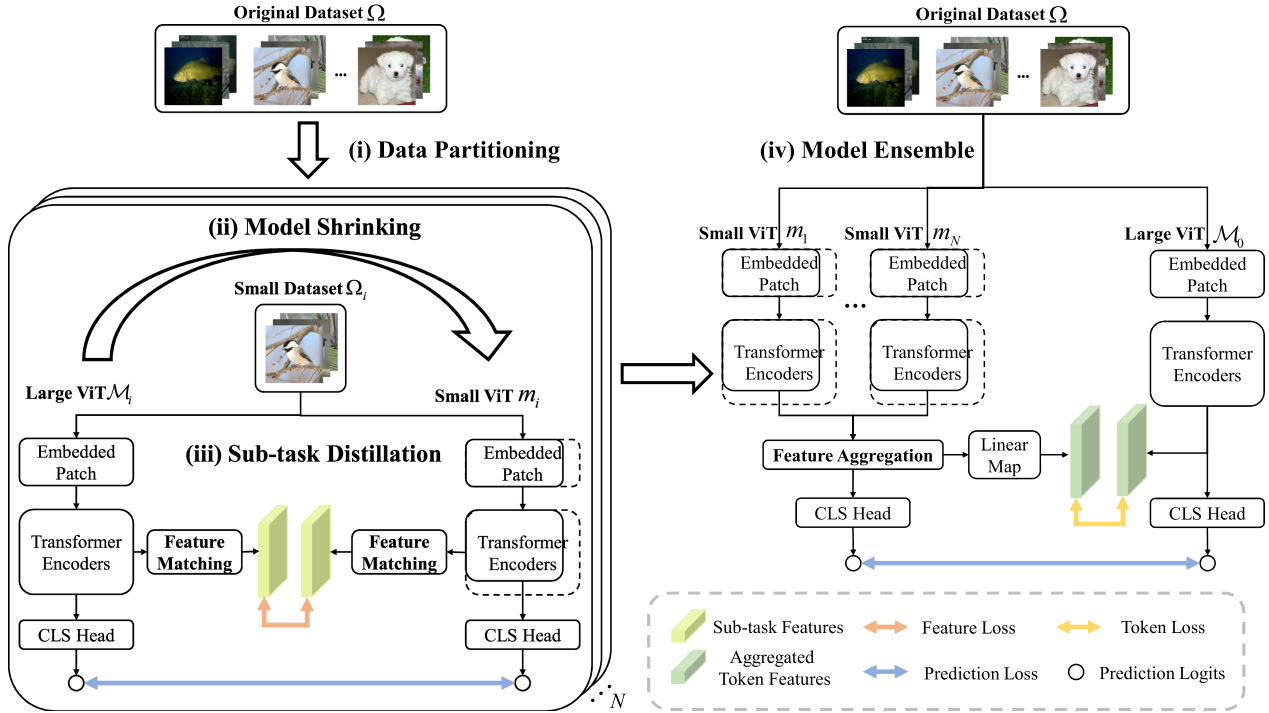
Fig. 4. Overview of the DEKD algorithm. The algorithm requires a dataset and multiple small datasets, utilizes multiple large ViT as teacher models and multiple small ViT as student models, as well as contains a feature matching module and aggregation module. The procedure includes four steps: (i) The original dataset is equally partitioned into N small datasets. The large ViT is also decomposed into N small models. (ii) By model shrinking, unimportant parts for specific classes in the small models can be removed. (iii) During sub-task distillation, the small models, as student, can learn fine-grained knowledge from the large teacher ViT by feature matching. (iv) The $[class]$ tokens from the small models are fused by the feature aggregation module. We also utilize the large ViT to improve the collaborative inference performance. The aggregated results are passed into the task-specific head to obtain the final results.

Discussion: The proposed collaborative deployment framework of ViTs on multiple edge devices is specially designed to achieve efficient deployment of ViTs so that the inference latency is low and accuracy is comparable to the powerful giant ViTs. We propose to decompose the single giant transformer model to multiple small models, and then deploy these small models on edge devices. In this way, only small amounts of data need to be transmitted and feature aggregation module is efficient. By decomposing ViT into models as small as possible and designing an efficient module to aggregate, we can significantly reduce the inference latency and save plenty of energy with only a slight accuracy loss.

IV. ALGORITHM DESIGN

In order to decompose the ViT into lightweight models and aggregate results of small models to obtain a satisfactory performance, we propose decomposition and ensemble based on knowledge distillation termed DEKD. In this section, we first provide an overview of our method, and then depict the details of each part.

A. Overview

An overview of our proposed DEKD algorithm is shown in Fig. 4. The algorithm requires a dataset and multiple small datasets, utilizes multiple large ViT as teacher models and multiple small ViT as student models, as well as contains a feature

matching module and aggregation module. Assume we have N edge devices, and the DEKD algorithm follows a four-step procedure as shown in Algorithm 1:

i) *Data partitioning:* We partition the original dataset into N small datasets according label classes.

ii) *Model shrinking:* We also decompose the large ViT into N small models. Every small model only preserves the structure which is more important to the specific small dataset than other small datasets.

iii) *Sub-task distillation:* We use N small datasets to train N large ViTs, which are randomly initialized, respectively. These pretrained large ViTs are regarded as teacher models. Each small model can learn the intermediate knowledge from the specific teacher model by utilizing the feature matching module.

iv) *Model ensemble:* We utilize the feature aggregation module to fuse the $[class]$ tokens from the trained small models. Moreover, intermediate knowledge of the large ViT trained on the original dataset can also be used as supervision to improve the aggregation performance.

B. Data Partitioning

We first introduce the proposed data partitioning approach as shown in Lines 1 to 4 of Algorithm 1. Due to their limited model capacity, the small models cannot finish complicated inference tasks with an acceptable accuracy and latency. Thus, we propose
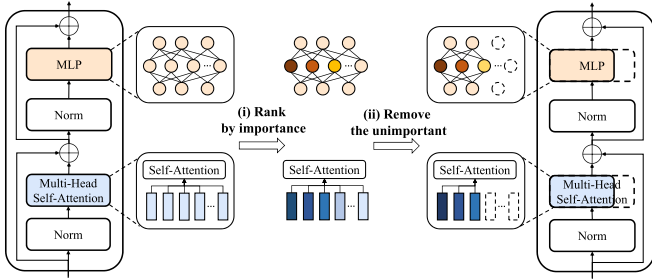
Fig. 5. Procedure of Model Shrinking. Firstly, the importance of each head in MSA and each neuron in the hidden layer of MLP is computed. Then we sort heads and neurons by importance. The unimportant parts for specific classes are then removed. Finally, connections of the remaining heads and neurons are reconstructed.

to decompose the original dataset into multiple small datasets to simplify the tasks. Here we take an $\mathcal{N}_{class}$-class classification task as an example.

First, the original dataset Ω_0 is equally decomposed into N small datasets according to the class label. The number of class in the dataset is denoted by $\mathcal{N}_{class}$. If the number of class $\mathcal{N}_{class}$ is divisible by the number of small datasets N, the number of class $\mathcal{N}_i$ in the small datasets is $\mathcal{N}_{class}/N$. Otherwise, the number of class $\mathcal{N}_i$ in the i-th small dataset is given by

$$\mathcal{N}_i = \begin{cases} \lfloor \mathcal{N}_{class}/N \rfloor + 1, & \text{if } i = 1, 2, \ldots, n \\ \lfloor \mathcal{N}_{class}/N \rfloor, & \text{if } i = n+1, \ldots, N \end{cases} \quad (4)$$

where $n = \mathcal{N}_{class}$ MOD N. Then, we randomly sample $\mathcal{N}_i$ classes of data from dataset Ω_0. These sampled data construct the small dataset Ω_i. Finally, after repeating sampling and dataset construction for N times, we obtain N partitioned small datasets.

C. Model Shrinking

When the small datasets are available, we next introduce the proposed model shrinking method. The ViT is decomposed into N small models by following the process in Section III-B. In order to ensure only slight performance loss and simultaneously reduce the computation cost, we only preserve the most important parts in the small models. In the multiheaded self-attention module, different heads can capture various semantic information [20]. Some researchers [18] also found that some neurons are responsible for specific classes. Thus, we can remove the unimportant heads in MSA and the unimportant neurons in the hidden layer of MLP to reduce the size and computation of the small models. The important parts for specific classes in the small models shall be reserved. We define the shrinking factor σ to dynamically adjust the model size according to hardware requirements. The shrinking factor indicates that the last $\sigma\%$ of heads and neurons shall be removed.

Fig. 5 illustrates the details of the proposed model shrinking approach. First, we use testing set of the small datasets partitioned from the original dataset to compute the importance of heads in the MSA module and of the neurons in the hidden layer of the MLP block. Then, we rank these heads and neurons by their importance and remove unimportant parts. Finally, the connections of the small models are reconstructed.

Algorithm 1: DEKD.

Notations: large ViT $\mathcal{M}_0$ trained on original dataset Ω_0; large ViTs $\{\mathcal{M}_1, \ldots, \mathcal{M}_N\}$ trained on small datasets $\{\Omega_1, \ldots, \Omega_N\}$, respectively; shrinking factor σ; In sub-task distillation, learning rates $\{\mu_1, \ldots, \mu_N\}$; first momentum factors $\{\rho_{1,1}, \ldots, \rho_{1,N}\}$; second momentum factors $\{\rho_{2,1}, \ldots, \rho_{2,N}\}$ weight decay factors $\{\lambda_1, \ldots, \lambda_N\}$; In model ensemble, learning rate μ; first momentum factor ρ_1; second momentum factor ρ_2; weight decay factor λ; the ensemble of small trained models m_0;

   /* (1) Data Partition   */
1 Compute the number of class $\mathcal{N}_i$ by (4);
2 **for** $i := 1$ *to* N **do**
3   Randomly sample $\mathcal{N}_i$ classes data from Ω_0;
4   Using sampled data to generate small dataset Ω_i;
     /* (2) Model Shrinking   */
5   Using small dataset Ω_i to train $\mathcal{M}_i$;
     /* (i) Rank by importance   */
6   Compute head importance $I_{h,i}$ for $\mathcal{M}_i$ by (6);
7   Compute neuron importance $I_{w,i}$ for $\mathcal{M}_i$ by (7);
8   Rank heads and neurons in descending order of importance;
     /* (ii) Remove unimportant parts   */
9   Remove last σ percent of heads and neurons;
10   Reconstruct the connections of the model and get new small model m_i;
   /* (3) Sub-task Distillation   */
11 **for** $i := 1$ *to* N **do**
12     $\boldsymbol{\theta}_i \leftarrow$ **Sub-task Distillation**$(\Omega_i, m_i, \mathcal{M}_i, \mu_i, \rho_{1,i}, \rho_{2,i}, \lambda_i)$
   /* (4) Model Ensemble   */
13 **Initialize** parameters for m_0 using $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N\}$;
14 **Initialize** parameters for large ViT $\mathcal{M}_0$ by training on original dataset Ω_0;
15 $\boldsymbol{\theta}_0 \leftarrow$ **Model Ensemble**$(\Omega_0, \boldsymbol{\theta}_0, \mu, \rho_1, \rho_2, \lambda, \mathcal{M}_0)$;

In order to identify the importance of heads and neurons in encoders for certain classes, we utilize the importance metric as in [20], [21]. The importance metric I of a head or a neuron can be computed by using the variation of training loss $\mathcal{L}$ if a head or a neuron is removed. For one head with output $\mathbf{O}_h$, the training loss $\mathcal{L}(\mathbf{O}_h)$ can be expressed as

$$\mathcal{L}(\mathbf{O}_h) = \mathcal{L}(\mathbf{O}_h^*) + \frac{\partial \mathcal{L}}{\partial \mathbf{O}_h}(\mathbf{O}_h - \mathbf{O}_h^*) + R_h, \quad (5)$$

by using Taylor expansion, where R_h is the remainder of the term that is smaller in magnitude than the previous terms if $\mathbf{O}_h$ is sufficiently close to $\mathbf{O}_h^*$. If we remove head h, $\mathbf{O}_h^* = 0$. Then, the importance of head h is

$$\begin{aligned} I_h &= |\mathcal{L}(\mathbf{O}_h) - \mathcal{L}(\mathbf{O}_h = 0)| \\ &= \left| \frac{\partial \mathcal{L}}{\partial \mathbf{O}_h} \mathbf{O}_h + R_h \right| \approx \left| \frac{\partial \mathcal{L}}{\partial \mathbf{O}_h} \mathbf{O}_h \right|, \end{aligned} \quad (6)$$

where the remainder R_h can be ignored. Similarly, the weights of a neuron connected with $\mathbf{W}_1$ and $\mathbf{W}_2$ in the MLP block are defined as $w = \{w_1, w_2, \ldots, w_{2D}\}$. Thus, the importance of the neuron is given by

$$I_w = \left| \frac{\partial \mathcal{L}}{\partial \mathbf{O}_w} \mathbf{O}_w \right| = \left| \sum_{i=1}^{2D} \frac{\partial \mathcal{L}}{\partial \mathbf{O}_{w_i}} \mathbf{O}_{w_i} \right|. \tag{7}$$

Discussion: We use the small datasets partitioned from the original dataset to find the important parts for certain classes in the decomposed small models. We can flexibly control the size by removing the unimportant parts in the MSA and MLP modules according to hardware requirements. Our method reduces the model size and redundant information at the same time to guarantees the trade-off between inference accuracy and latency. We summarized the procedure in Lines 5 to 10 of Algorithm 1.

In order to minimize accuracy loss caused by decomposition and shrinking, we propose the sub-task distillation approach to transfer knowledge for specific classes from large ViTs to small decomposed models. The approach requires multiple small datasets, and contains multiple teachers and students as shown in the lower left part of Fig. 4. The teachers $\{\mathcal{M}_1, \mathcal{M}_2 \ldots, \mathcal{M}_N\}$ are large ViTs trained on small datasets, respectively. The students are decomposed small models m_i. A small dataset has correspondence between a pair of teacher and student. The student can learn knowledge for specific classes with the help of the teacher trained on the corresponding small dataset. During distillation, the predictions and intermediate features in teachers are chosen to transfer. For the sake of intermediate distillation, we utilize a feature matching module to match the dimensions of teachers and students. In the following, we introduce the feature matching module, and then present the objective function for distilling the knowledge of large ViTs.

Feature Matching: The proposed feature matching module $FM(\cdot)$ applies transformation in intermediate features to promote distillation when the feature dimensions of teachers and students are different. Recent works found that transferring knowledge base on features [22] can be utilized to help the training of the student. However, these methods usually have restrictions on mode layers [23] or significant information loss when performing dimension matching [24]. This motivates us to design a novel feature matching module to break the restriction and reduce the performance loss in distillation.

Details of our feature matching module are shown in Fig. 6. We suppose the feature map from teachers is $X_t \in \mathbb{R}^{a \times b \times c}$, and the feature map from students is $X_s \in \mathbb{R}^{a \times b \times c'} (c' \leq c)$. Firstly, the feature maps are expanded to be of three dimensional. By matricizing teacher and student according to each of their three dimensions, we can get six matrices $X_{t_1} \in \mathbb{R}^{a \times bc}, X_{t_2} \in \mathbb{R}^{b \times ac}, X_{t_3} \in \mathbb{R}^{c \times ab}, X_{s_1} \in \mathbb{R}^{a \times bc'}, X_{s_2} \in \mathbb{R}^{b \times ac'}$, and $X_{s_3} \in \mathbb{R}^{c' \times ab}$. Then, we utilize the function $f(X) = XX^T$ to match the second dimension of X_{t_i} and X_{s_i}. But the numbers of the first dimension between X_{t_3} and X_{s_3} are different. Hence, before applying the function f, X_{t_3} needs to be randomly sampled to get c' rows to construct a new matrix X'_{t_3}. In this way, our feature matching module can break the restriction
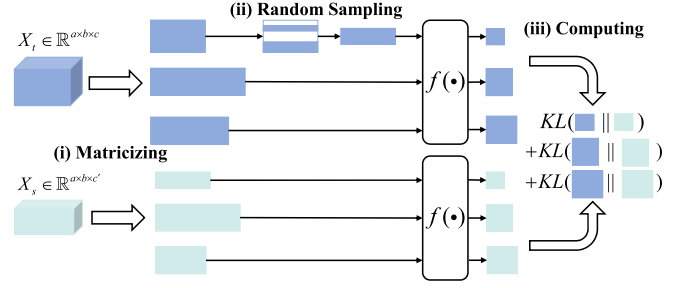

Fig. 6. Feature matching module in sub-task distillation. The 3D features from the teacher and student are matricized to be 2D matrices with different dimensions. If dimensions of the matrices between the teacher and student are not the same, the matrix from the teacher is randomly sampled to get the same number of rows as the matrix of the student. Then these matrices are transferred to square matrices with the same dimension by $f(\cdot)$. Finally, KL divergence can be used to calculate their differences.

Algorithm 2: Sub-task Distillation.

Input: partitioned small dataset Ω_i; small model m_i; large ViT models $\mathcal{M}_i$; learning rates μ_i; first momentum factors $\rho_{1,i}$; second momentum factors $\rho_{2,i}$; weight decay factors λ_i;

Output: model m_i with optimized parameters θ_t;

1 **Initialize** the parameter $\theta_{t=0}$ of model m_i by training in pretraining dataset;

2 **Initialize** time step $t \leftarrow 0$, first moment vector $\boldsymbol{u}_{t=0} \leftarrow \mathbf{0}, \epsilon \leftarrow 10^{-8}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \mathbf{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}^n$; max training steps $I_{max} \leftarrow 10^5$;

3 **for** $t := 1$ to I_{max} **do**

4   Select batch from Ω_i;

5   Apply $FM(\cdot)$ transformation in $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ vectors;

6   Compute $\mathcal{L}_{\text{feat}}$ with (12);

7   Compute $\mathcal{L}_{\text{sub-task}}$ with (8);

8      $g_t \leftarrow \mathcal{L}_{\text{sub-task}}(\theta_{t-1}) + \lambda_i \theta_{t-1}$;

9      $\boldsymbol{u}_t \leftarrow \rho_{1,i} \boldsymbol{u}_{t-1} + (1 - \rho_{1,i}) \boldsymbol{g}_t$;

10      $\boldsymbol{v}_t \leftarrow \rho_{2,i} \boldsymbol{v}_{t-1} + (1 - \rho_{2,i}) \boldsymbol{g}_t^2$;

11      $\hat{\boldsymbol{u}}_t \leftarrow \boldsymbol{u}_t / (1 - \rho_{1,i}^t)$;

12      $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \rho_{2,i}^t)$;

13      $\eta_t \leftarrow SetScheduleMultiplier(t)$;

14      $\theta_t \leftarrow \theta_{t-1} - \eta_t (\mu_i \hat{\boldsymbol{u}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \theta_{t-1})$;

15   **if** *stopping criterion is met* **then**

16   **Output** m_i with optimized parameters θ_t;

17   **break**;

---

of feature dimension difference and reduce computation cost during feature matching.

D. Sub-Task Distillation

Feature and Prediction Distillation: We use the predictions and intermediate features of teachers to guide the training of students. The student can learn knowledge from the teacher by minimizing the following objective:

$$\mathcal{L}_{\text{sub-task}} = \mathcal{L}_{pred} + \beta \mathcal{L}_{feat}, \tag{8}$$

where β is a balancing hyperparameter.

For distillation based on predictions, we follow [14] to take the hard decision of the teacher as a true label. The prediction logits of teacher and student are Y_t and Y_s, respectively. Different from the traditional hard label in knowledge distillation, we utilize the decision of teacher as the label:

$$y_t = \arg\max_s Y_t(s), \qquad (9)$$

where s represents all categories in the training set, and y_t plays the same role as true label y, but with a better performance than y. The hard decision of teacher may change depending on the specific data augmentation. The student can learn more knowledge with data augmentation especially for the ViTs which require extensive data augmentations to guarantee performance. Hence, using y_t is a better choice. The objective of prediction distillation is

$$\mathcal{L}_{pred} = \frac{1}{2}\mathcal{L}_{CE}\left(\phi(Y_s), y\right) + \frac{1}{2}\mathcal{L}_{CE}\left(\phi(Y_s), y_t\right), \qquad (10)$$

where $\phi(\cdot)$ represents the prediction results and $\mathcal{L}_{CE}$ is the cross entropy loss, given by

$$\mathcal{L}_{CE}(p, q) = -\sum_{i=1} p_i \log(q_i). \qquad (11)$$

In the distillation based on intermediate features, we utilize the $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ vectors in MSA modules to supervise the training of students. Recently, it is found that attention mechanism can be utilized to capture rich semantic information [16]. Since the $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ vectors are the key parts of MSA module, we utilize these vectors as the supervision of students to learn the semantic information of teachers. The dimensions of the three vectors are h, M and 64, respectively, where h is the number of heads and M is the number of tokens. The number of heads and tokens are different between teachers and students due to the model decomposition and shrinking. Hence, the feature matching module $FM(\cdot)$ is utilized to match their dimensions. For notational simplicity, we denote $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ as $\mathbf{P}_1, \mathbf{P}_2$, and $\mathbf{P}_3$, respectively. The distillation objective is represented as

$$\mathcal{L}_{feat} = \frac{1}{3l} \sum_{j,k=1}^{l} \sum_{i=1}^{3} \alpha_i \mathcal{L}_{KL}\left(FM(\mathbf{P}_{i,s}^j) || FM(\mathbf{P}_{i,t}^k)\right), \qquad (12)$$

$$\mathcal{L}_{KL}(p||q) = \sum_{i=1} q_i \log \frac{q_i}{p_i}, \qquad (13)$$

where α_i is a balancing hyperparameter and $\mathcal{L}_{KL}$ is the KL-divergence loss to help learn the distribution of target samples. Superscripts j and k signify the j-th layer of the student and the k-th layer of the teacher, respectively. The student model has j layers, and we use a linear mapping function $g(\cdot)$ to select the layer of teachers to transfer, i.e., $k = g(j)$. For example, the j-th layer of the student model can learn knowledge from the $g(j)$-th layer of the teacher. By utilizing the mapping function for layers, we can transfer knowledge from teachers to the students layer-to-layer.

Algorithm 3: Model Ensemble.

Input: dataset Ω_0; the ensemble of small trained models m_0 with its parameters $\boldsymbol{\theta}_0$; learning rate μ; first momentum factor ρ_1; second momentum factor ρ_2; weight decay factor λ; teacher model $\mathcal{M}_0$;

Output: model m_0 with optimized parameters $\boldsymbol{\theta}_t$;

1 **Initialize** model m_0 parameter $\boldsymbol{\theta}_{t=0} \leftarrow \boldsymbol{\theta}_0$, time step $t \leftarrow 0$, $\epsilon \leftarrow 10^{-8}$, first moment vector $\boldsymbol{u}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}^n$; max training steps $I_{max} \leftarrow 10^5$;

2 **for** $t := 1$ *to* I_{max} **do**

3   Select batch from Ω_0;

4   Utilize $FA(\cdot)$ to aggregate CLS tokens with (3);

5   Compute $\mathcal{L}_{token}$ with (15);

6   Compute $\mathcal{L}_{ensemble}$ with (14);

7     $\boldsymbol{g}_t \leftarrow \mathcal{L}_{\text{sub-task}}(\boldsymbol{\theta}_{t-1}) + \lambda_i \boldsymbol{\theta}_{t-1}$;

8     $\boldsymbol{u}_t \leftarrow \rho_{1,i}\boldsymbol{u}_{t-1} + (1 - \rho_{1,i})\boldsymbol{g}_t$;

9     $\boldsymbol{v}_t \leftarrow \rho_{2,i}\boldsymbol{v}_{t-1} + (1 - \rho_{2,i})\boldsymbol{g}_t^2$;

10    $\hat{\boldsymbol{u}}_t \leftarrow \boldsymbol{u}_t/(1 - \rho_{1,i}^t)$;

11    $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1 - \rho_{2,i}^t)$;

12    $\eta_t \leftarrow SetScheduleMultiplier(t)$;

13    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\mu_i \hat{\boldsymbol{u}}_t/(\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1}\right)$;

14   **if** *stopping criterion is met* **then**

15   **Output** m_0 with optimized parameters $\boldsymbol{\theta}_t$;

16   **break**;

Discussion: The proposed sub-task distillation method can help to transfer knowledge for specific classes from large ViT to decomposed small models. The predictions and intermediate features from teachers can promote the small models to learn more fine-grained knowledge. This can substantially reduce the performance drop caused by decomposition and shrinking. The sub-task distillation procedure is summarized in Algorithm 2.

E. Model Ensemble

After the distillation of small models using corresponding small datasets, we resort to a distillation-based model ensemble method to improve the collaborative inference of multiple small models. In this subsection, we depict the method and then formulate the objective function for training the multiple small models.

The architecture of the method is shown in the right part of Fig. 4. The method is composed of a teacher and a student, and uses a dataset Ω_0. The teacher $\mathcal{M}_0$ is the large ViT trained on the original dataset Ω_0. The student is the ensemble of multiple decomposed small models trained by sub-task distillation. Based on knowledge distillation, the large ViT can contribute to the ensemble of multiple small models to achieve a good collaborative inference performance.

Token and Prediction Distillation: We use the $[class]$ token and predictions from the teacher for supervision of the student. The objective function is given by

$$\mathcal{L}_{ensemble} = \mathcal{L}_{pred} + \gamma \mathcal{L}_{token}, \qquad (14)$$

where γ is a balancing hyperparameter. $\mathcal{L}_{pred}$ and $\mathcal{L}_{token}$ are the objective functions for prediction-based and token-based distillation, respectively. Different from Section IV-D where student is the single small model, the student here is composed of multiple small models which are responsible for specific classes, respectively. Each small model can extract different features, and thus it is not appropriate to utilize the same intermediate features to supervise the student training as these in Section IV-D. In the ViT model, the $[class]$ token, including target semantic information, is used to predict the class. Hence, considering the trade-off between ensemble performance and complexity, we aggregate the $[class]$ tokens of all small models. The objective function of token distillation is

$$\mathcal{L}_{token} = \mathcal{L}_{MSE}\left(FA(Z_s)G, Z_t\right), \tag{15}$$

where $Z_s = \{Z_{s_1}, Z_{s_2}, \ldots, Z_{s_N} \in \mathbb{R}^{1 \times D_s}\}$ is the set of $[class]$ tokens of all small models, $Z_t \in \mathbb{R}^{1 \times D_t}$ is the $[class]$ token of the teacher, and the matrix $G \in \mathbb{R}^{ND_s \times D_t}$ is a learnable linear map, which transforms the aggregated $[class]$ tokens of all small models into the space of the teacher. The feature aggregation module $FA(\cdot)$ has the same structure as the aggregation part in Section III-B. The mean squared loss $\mathcal{L}_{MSE}$ is utilized to represent the differences between the aggregated tokens and the token from the teacher. The objective function of prediction distillation is the same as the one adopted in (10).

Discussion: We propose a distillation-based model ensemble method to fuse multiple small models and improve the collaborative inference performance. By using both token and prediction distillation, we guarantee slight accuracy loss compared to large ViT. The procedure of Model Ensemble is illustrated in Algorithm 3.

V. EXPERIMENTS

In this section, we first introduce some implementation details and experiment settings. Then we evaluate our methods on GPU servers and edge devices, respectively. Finally, we conduct extensive ablation experiments to demonstrate the superiority of our methods.

A. Implementation Details

In this subsection, we first introduce prototype and deployment details. Then we depict the datasets, backbones, and counterparts, followed by evaluation metrics.

Prototype: We implement DeViT on a real-world collaborative edge computing testbed that consists of 4 NVIDIA Jetson Nano devices [25] and 1 switch. Fig. 7 shows the implemented hardware platform for DeViT. The Jetson Nano contains a Quad-core ARM Cortex-A57 MPCore processor at 1.6 GHz, a 4 GB RAM Memory, which represents the weak edge device. These edge devices are connected via a gigabyte switch TP-LINK TL-SG1008D. For bandwidth control, we use the traffic control tool tc [26], which is able to limit the bandwidth under the setting value. The maximum bandwidth between devices are fixed at 2 MB/s.

Deployment: We use PyTorch as the backend engine to execute ViT models. The employed ViT models are trained
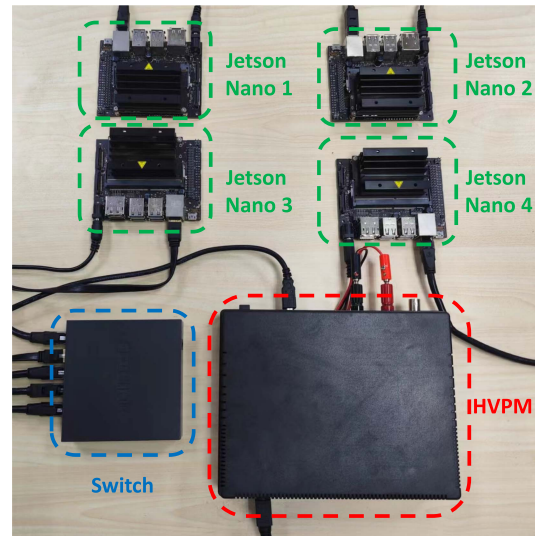

Fig. 7. Our experimental prototype employs four Jetson Nano and one switch. We utilize the Monsoon High Voltage Power Monitor (HVPM) to measure the energy and power.

following DEKD algorithm and deployed on all devices in advance. The communication module is implemented based on gPRC [27]. All the compared approaches are run with timm [28], a GitHub repository, for fair comparison.

Datasets: We evaluate the proposed methods on four public computer vision datasets: CIFAR-100 [29], Oxford 102 Flower [30], Stanford Cars [31], and ImageNet-1K [32]. All images are resized to be 224 × 224 in our experiment. The details of each dataset are given as follows:

- *CIFAR-100* consists of 60,000 images that belong to 100 classes uniformly. In each class, there are 500 images for training and 100 images for testing.
- *Oxford 102 Flower* is an image classification dataset consisting of 102 flower categories. The flowers are common in the United Kingdom. Each class consists of 40 to 258 images.
- *Stanford Cars* dataset consists of 196 classes of cars with a total of 16,185 images, taken from the rear angle. The dataset is divided into almost equal train/test subsets with 8,144 training images and 8,041 testing images.
- *ImageNet-1 K* is a large-scale hand-labeled image recognition dataset consisting of 1,000 categories. There are 1.2 million samples in the training set and 50,000 samples in the testing set.

Backbones: The configurations of different backbones are listed in Table II. We show the large models and their corresponding small models decomposed by our method. In the table, #1, #3, and #5 represent the large ViTs. #2, #4, and #6 show the decomposed small models using our method. We use the following models as backbones:

- *ViT [5]:* The first pure transformer structure for computer vision tasks.
- *DeiT [14]:* DeiT adds a distillation token to learn knowledge of the teacher, as compared to ViT.

TABLE II
CONFIGURATION SETTINGS OF DIFFERENT BACKBONES UTILIZED IN OUR
METHOD

#	Model	#Layers	Embedding dimension	MLP size	#Heads
1	ViT-L/16	24	1024	4096	16
2	**DeViT**	12	192	768	3
3	DeiT-B	12	768	3072	12
4	**DeDeiT**	12	192	768	3
5	CCT-14_7×2	14	384	1152	6
6	**DeCCT**	7	256	512	4

- *CCT [33]:* CCT is a CNN+Transformer hybrid model that uses CNNs and sequence pooling to replace patch embeddings and class token.

Counterparts: We first compare the large ViTs for aforementioned three backbones, including ViT-L/16, DeiT-B and CCT-14_7 2, to demonstrate the effectiveness of our methods. Then we compare our methods with four representative lightweight ViT approaches to show the superiority.

- *T2T-ViT$_t$-14 [34]* incorporates a layer-wise transformation to structuralize the image to tokens by recursively aggregating neighboring tokens.
- *Twins-PCPVT-S [35]* introduces a carefully-designed yet simple spatial attention mechanism.
- *Twins-SVT-S [35]* introduces locally-grouped self-attention and global sub-sampled attention to capture fine-grained and global information.
- *MobileViT-S [12]* designs a light-weight ViT combining CNNs for mobile devices. Note that we employ our ensemble method to aggregate original MobileViT-S for fair comparison, denoted as MobileViT-S-Ens.

Evaluation Metrics: We employ the Top-1 classification accuracy as the performance metric. The end-to-end latency, energy consumption and average power are utilized as the efficiency metrics. As the models are deployed, we use the PyTorch Profiler tool [36] to profile the end-to-end latency of one inference. The whole energy and power are measured by Monsoon High Voltage Power Monitor [37], which can collect the output voltage and current at a sampling rate of 5,000 samples per second. Before measurement, we switch the NVIDIA Jetson Nano to the max power mode, i.e., the max power is 10 Watt, and turn off the dynamic voltage and frequency scaling to ensure a steady measurement environment. We perform inference on the test set of the ImageNet dataset using NVIDIA Jetson Nanos, and measure the inference overhead of 50,000 testing samples in 1,000 different categories. For each ViT model, we run it for once as a warm-up and then record the execution time with 100 runs without break for the whole testing set. The aim of warm-up running is to alleviate the impact of weight loading and PyTorch initiation since we have omitted these overheads. We minus the power of the standby system from the measured power to acquire the power of performing inference. Thus, the energy consumption of a single inference can be obtained by taking the product of the averaged latency and the power of performing inference.

B. Evaluation on the GPU Server

In this set of experiments, we measure the classification accuracy results on four widely-used datasets as well as end-to-end inference latency on the GPU server. We choose ViT-L/16, DeiT-B, and CCT-14_7 2 as large ViTs to be decomposed. These models are all decomposed into 4 small models. End-to-end latency is measured on a GPU server with NVIDIA RTX 3090 GPUs, which is the average value with 100 runs without break for the whole testing set of corresponding datasets. We present the results compared with ViTs of different model sizes as follows.

Comparison With Large ViTs: We compare our methods with the large ViTs for various ViT backbones in the GPU server. The results of accuracy and end-to-end latency on four datasets are shown in Table III. We can find that large ViTs achieve superior performance nevertheless with extremely high latency overheads and huge computation costs, while our methods can reduce computation costs and accelerate inference latency by 1.98 × on average with only a 2% accuracy drop. In most real-world deployments, a 2% accuracy loss is considered negligible given the considerable latency reduction. Specifically, DeViTs can run 2.89× faster with only 0.96% accuracy sacrifice than ViT-L/16. Our methods can also accelerate end-to-end latency by approximately 1.99 × with a 1.55% accuracy drop on average for different backbones even on the large-scale dataset, ImageNet-1 K. These results demonstrate that our framework can reduce the computation cost of large ViTs for different backbones. Moreover, our models achieve real-time inference latency with only slight accuracy sacrifice on the GPU server.

Comparison With Lightweight ViTs: We compare our methods to a variety of representative baseline models with approximate model parameters. Table IV presents the results of Top-1 classification accuracy and end-to-end inference latency on four datasets. We can observe that the DeCCTs significantly outperforms all other lightweight ViTs with the lowest latency and acceptable accuracy. Specifically, DeViTs/DeDeiTs/DeCCTs can reduce inference latency by 34.47%/31.49%/42.77%, respectively, compared with the state-of-the-art efficient ViT model, MobileViT-S, on ImageNet-1 K, which shows that our methods achieve an excellent trade-off between efficiency and performance. On the other hand, we can also find that although the parameters or FLOPs of some models (i.e., Twins-SVT-S, Twins-PCPCT-S, and MobileViT-S) are less than that of our models, their end-to-end latency results on the GPU server are not lower than our models. It suggests that the number of parameters or FLOPs is only indicative but cannot fully determine the inference efficiency, which may not be an appropriate criterion for inference latency.

C. Evaluation on the Edge Device

In this subsection, we evaluate our methods on the edge devices and measure the Top-1 classification accuracy, end-to-end inference latency, average energy consumption, and average power to compare our methods with large ViTs and other lightweight ViT models. We deploy these models on the NVIDIA Jetson Nanos and employ an external high voltage power monitor [37] for the measurement. We first present the

TABLE III
COMPARISON BETWEEN OUR METHODS AND LARGE ViTs IN THE GPU SERVER USING DIFFERENT DATASETS

Model	#Params.	FLOPs	Results: CIFAR-100		Results: Stanford Cars		Results: 102 Flowers		Results: ImageNet-1K	
			Acc. ↑	Latency ↓	Acc. ↑	Latency ↓	Acc. ↑	Latency ↓	Acc. ↑	Latency ↓
ViT-L/16	304.0 M	190.7 G	88.77 %	$18.65_{\pm 0.5}$ ms	89.83 %	$18.68_{\pm 0.8}$ ms	98.13 %	$18.19_{\pm 1.5}$ ms	76.50 %	$18.57_{\pm 0.7}$ ms
DeViTs	23.0 M	5.9 G	87.98 %	$\mathbf{6.45}_{\pm 0.3}$ ms	89.11 %	$\mathbf{6.25}_{\pm 0.7}$ ms	96.73 %	$\mathbf{6.50}_{\pm 0.2}$ ms	74.85 %	$\mathbf{6.16}_{\pm 0.3}$ ms
DeiT-B	86.6 M	35.1 G	90.80 %	$8.51_{\pm 0.2}$ ms	92.10 %	$8.62_{\pm 0.8}$ ms	98.40 %	$8.39_{\pm 0.1}$ ms	83.40 %	$8.32_{\pm 0.7}$ ms
DeDeiTs	23.9 M	5.9 G	89.34 %	$\mathbf{6.62}_{\pm 0.3}$ ms	91.53 %	$\mathbf{6.51}_{\pm 0.4}$ ms	97.89 %	$\mathbf{6.76}_{\pm 0.6}$ ms	81.94 %	$\mathbf{6.44}_{\pm 0.3}$ ms
CCT-14_7×2	22.4 M	11.1 G	88.78 %	$8.56_{\pm 0.2}$ ms	91.78 %	$9.11_{\pm 0.3}$ ms	97.97 %	$10.16_{\pm 1.1}$ ms	80.67 %	$8.96_{\pm 0.1}$ ms
DeCCTs	18.3 M	5.1 G	87.43 %	$\mathbf{5.16}_{\pm 0.2}$ ms	90.67 %	$\mathbf{5.35}_{\pm 0.3}$ ms	96.84 %	$\mathbf{5.30}_{\pm 0.1}$ ms	79.12 %	$\mathbf{5.38}_{\pm 0.2}$ ms

TABLE IV
COMPARISON BETWEEN OUR METHODS AND LIGHTWEIGHT BASELINE MODELS IN THE GPU SERVER USING DIFFERENT DATASETS

Model	#Params.	FLOPs	Results: CIFAR-100		Results: Stanford Cars		Results: 102 Flower		Results: ImageNet-1K	
			Acc. ↑	Latency ↓	Acc. ↑	Latency ↓	Acc. ↑	Latency ↓	Acc. ↑	Latency ↓
T2T-ViT$_t$-14	21.5 M	12.2 G	86.62 %	$9.89_{\pm 0.8}$ ms	90.98 %	$10.38_{\pm 0.9}$ ms	97.81 %	$9.64_{\pm 0.1}$ ms	81.70 %	$9.84_{\pm 0.7}$ ms
Twins-SVT-S	24.0 M	2.9 G	86.72 %	$13.79_{\pm 0.9}$ ms	91.08 %	$14.84_{\pm 1.3}$ ms	97.82 %	$15.78_{\pm 1.5}$ ms	81.70 %	$15.18_{\pm 1.4}$ ms
Twins-PCPVT-S	24.1 M	3.8 G	86.70 %	$13.13_{\pm 1.3}$ ms	91.46 %	$14.33_{\pm 1.3}$ ms	**97.98** %	$13.01_{\pm 0.1}$ ms	81.20 %	$13.85_{\pm 1.0}$ ms
MobileViT-S	5.0 M	2.0 G	87.00 %	$9.22_{\pm 0.7}$ ms	89.99 %	$9.49_{\pm 0.2}$ ms	97.63 %	$9.29_{\pm 0.1}$ ms	78.40 %	$9.40_{\pm 0.1}$ ms
MobileViT-S-Ens	20.1 M	8.1 G	85.80 %	$9.72_{\pm 0.6}$ ms	88.12 %	$9.81_{\pm 0.3}$ ms	96.01 %	$9.96_{\pm 0.2}$ ms	78.62 %	$9.98_{\pm 0.2}$ ms
DeViTs	23.0 M	5.9 G	87.98 %	$6.45_{\pm 0.3}$ ms	89.11 %	$6.25_{\pm 0.7}$ ms	96.76 %	$6.50_{\pm 0.2}$ ms	74.85 %	$6.16_{\pm 0.3}$ ms
DeDeiTs	23.9 M	5.9 G	**89.34** %	$6.62_{\pm 0.3}$ ms	**91.53** %	$6.51_{\pm 0.4}$ ms	97.89 %	$6.76_{\pm 0.6}$ ms	**81.94** %	$6.44_{\pm 0.3}$ ms
DeCCTs	18.3 M	5.1 G	87.43 %	$\mathbf{5.16}_{\pm 0.2}$ ms	90.67 %	$\mathbf{5.35}_{\pm 0.3}$ ms	96.84 %	$\mathbf{5.30}_{\pm 0.1}$ ms	79.12 %	$\mathbf{5.38}_{\pm 0.2}$ ms

TABLE V
COMPARISON OF ON-DEVICE EVALUATION BETWEEN OUR METHODS AND LARGE ViTs USING IMAGENET-1 K DATASETS

Model	#Params.	FLOPs	Accuracy ↑	Latency ↓	Energy ↓	Power ↓
ViT-L/16	304.0 M	190.7 G	76.50 %	-	-	-
DeViTs	23.0 M	5.9 G	74.85 %	$\mathbf{31.30}_{\pm 2.3}$ ms	$\mathbf{204.29}_{\pm 25.0}$ mJ	$\mathbf{6.46}_{\pm 0.6}$ W
DeiT-B	86.6 M	35.1 G	83.40 %	-	-	-
DeDeiTs	23.9 M	5.9 G	81.84 %	$\mathbf{32.22}_{\pm 2.1}$ ms	$\mathbf{221.08}_{\pm 24.0}$ mJ	$\mathbf{6.86}_{\pm 0.7}$ W
CCT-14_7×2	22.4 M	11.1 G	80.67 %	$44.59_{\pm 4.9}$ ms	$432.50_{\pm 51.8}$ mJ	$9.69_{\pm 0.3}$ W
DeCCTs	18.3 M	5.1 G	79.12 %	$\mathbf{20.66}_{\pm 1.6}$ ms	$\mathbf{165.09}_{\pm 17.5}$ mJ	$\mathbf{7.99}_{\pm 0.6}$ W

results of comparison with large ViTs and then compare our methods with lightweight ViTs.

Comparison With Large ViTs: We compare our methods with the large ViTs for different ViT backbones on the edge device. Table V presents on-device comparison results on ImageNet-1 K. ViT-L/16 and DeiT-B are infeasible for deployment on resource-constrained platforms like Jetson Nano due to their substantial memory requirements and computational costs, denoted by "-" in their latency, energy, and power results. Despite the superior performance of larger ViTs, such as ViT-L/16 and DeiT-B, they remain nonviable for deployment on lower-performance edge devices. Our methods decompose the large ViTs for different backbones into multiple small models, facilitating efficient collaborative inference with accuracy nearly equivalent to the large ViTs. Specifically, our DeCCTs model improves inference latency by 2.16× and decreases energy consumption by 61.83% compared with CCT-14_7×2. These results demonstrate that our methods can alleviate the deploy-

ment of large ViTs, significantly accelerating inference speed and reducing energy consumption with acceptable accuracy.

Comparison With Lightweight ViTs: We compare our methods with other representative lightweight ViTs on the edge device. Table VI reports the on-device comparison results on ImageNet-1 K. We can observe that our methods can achieve the lowest end-to-end inference latency and energy consumption with approximate accuracy while maintaining comparable accuracy with lightweight ViTs. For example, DeCCTs improves inference latency by 60.92%, decreases energy consumption by 66.61%, and improves accuracy by 0.72% compared with the state-of-the-art lightweight ViT, MobileViT-S. Moreover, akin to the evaluation analysis on the GPU server, their end-to-end latency and energy consumption on real edge devices do not outperform our models despite the fewer number of parameters and FLOPs. Thus, there is no apparent correlation between the number of parameters or FLOPs and end-to-end latency and energy consumption of ViTs. The FLOPs and the number of

TABLE VI
COMPARISON OF ON-DEVICE EVALUATION BETWEEN OUR METHODS AND LIGHTWEIGHT BASELINE MODELS USING IMAGENET-1 K DATASETS

| Model	#Params.	FLOPs	Accuracy ↑	Latency ↓	Energy ↓	Power ↓
T2T-ViT$_t$-14	21.5 M	12.2 G	81.70 %	44.66$_{\pm2.8}$ ms	422.51$_{\pm33.8}$ mJ	9.46$_{\pm0.4}$ W
Twins-SVT-S	24.0 M	2.9 G	81.70 %	70.10$_{\pm5.5}$ ms	661.71$_{\pm57.5}$ mJ	9.44$_{\pm0.3}$ W
Twins-PCPCT-S	24.1 M	3.8 G	81.20 %	66.77$_{\pm4.8}$ ms	624.35$_{\pm53.6}$ mJ	9.35$_{\pm0.3}$ W
MobileViT-S	5.0 M	2.0 G	78.40 %	55.42$_{\pm5.6}$ ms	494.41$_{\pm50.9}$ mJ	8.92$_{\pm0.1}$ W
MobileViT-S-Ens	20.1 M	8.1 G	78.62 %	57.84$_{\pm6.4}$ ms	546.26$_{\pm53.5}$ mJ	9.42$_{\pm0.2}$ W
DeViTs	23.0 M	5.9 G	74.85 %	31.30$_{\pm2.3}$ ms	204.29$_{\pm25.0}$ mJ	**6.46$_{\pm0.6}$ W**
DeDeiTs	23.9 M	5.9 G	**81.94** %	32.22$_{\pm2.1}$ ms	221.08$_{\pm24.0}$ mJ	6.86$_{\pm0.7}$ W
DeCCTs	18.3 M	5.1 G	79.12 %	**20.66$_{\pm1.6}$ ms**	**165.09$_{\pm17.5}$ mJ**	7.99$_{\pm0.6}$ W
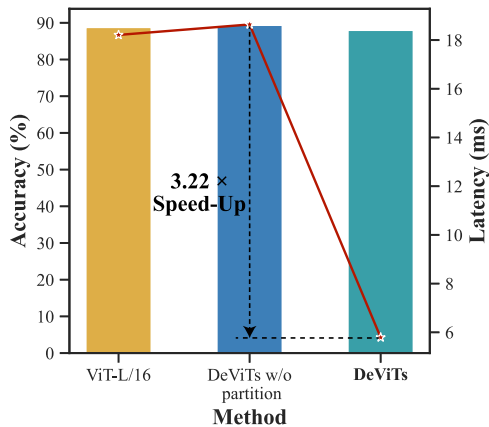
Fig. 8. Ablation study of dataset partition on CIFAR-100. "DeViTs w/o partition" represents utilizing the original dataset.

TABLE VII
ABLATION STUDY OF DIFFERENT LOSS FUNCTIONS IN SUB-TASK DISTILLATION

Model	GT	$\mathcal{L}_{pred}$	$\mathcal{L}_{feat}$	Average Acc.
ViT-L/16 (Teacher)	✓	✗	✗	92.92 %
DeViT-Sub w/o distillation	✓	✗	✗	85.92 %
DeViT-Sub (Student)	✗	✓	✗	87.20 %
	✗	✗	✓	89.57 %
	✗	✓	✓	**90.15 %**

parameters are not appropriate criteria to estimate the inference latency and energy consumption.

D. Ablation Study

To fully understand the impact of each part of the proposed framework, we design ablation studies, where all experiments are evaluated on CIFAR-100 and utilize ViT as the backbone. We first evaluate the effectiveness of partitioning datasets to find the important parts of ViTs and study the impact of different loss functions in the sub-task distillation stage. Then we analyze the superiority of our ensemble methods by comparing different ensemble strategies and study the effect of different device numbers.

Evaluation of Dataset Partition: We design an ablation experiment on CIFAR-100 to validate the effectiveness of utilizing the small partitioned datasets to find the important parts. The bar plot and line plot represent the results of accuracy and latency in Fig. 8, respectively. ViT-L/16 is the backbone model and decomposed into four small models. We can find that utilizing the small datasets as the proxy to find the important parts can reduce redundant computation to accelerate inference by 3.22× with only a 1.73 % accuracy drop compared to utilizing the original dataset. The results significantly validate the effectiveness of partitioning datasets and utilizing the small dataset as the proxy to find the important parts in decomposed models.

Evaluation of Sub-Task Distillation: The ablation study results of sub-task distillation are shown in Table VII. In this part, we choose the ViT-L/16 trained on different small partitioned datasets as the teacher for the corresponding small models. The backbone of each model is ViT. In the table, DeViT represents the decomposed small models for the backbone of ViT. GT means that the ground truth of the dataset is used as labels and distillation is not utilized. The Top-1 accuracy is the average results in the test set of all small partitioned dataset. From the results, we find that the small and shallow models can not learn deep and rich knowledge by themselves. With the help of large and powerful models, small models as students can learn the rich information effectively and acquire large performance improvement by distillation. Even if we only use predictions of the teacher for supervision, the average Top-1 accuracy can be enhanced by 1.28%. The results of using $\mathcal{L}_{feat}$ indicates that distillation based on intermediate features can promote small models to learn more fine-grained knowledge and improve classification performance. These two loss function $\mathcal{L}_{pred}$ and $\mathcal{L}_{feat}$ are both helpful, and the best result is achieved when both terms are adopted. The improvement is 4.23% compared to training without distillation.

Evaluation of Model Ensemble: In this part, we evaluate the performance of the model ensemble stage. Ablation study results under the supervision of powerful teacher models are shown in Table VIII. ViT-L/16 is a large powerful ViT and decomposed into four small models. It is also the teacher that guides the ensemble of all small decomposed models. Without the help of distillation, the direct ensemble of ViTs can lead to large accuracy loss. Compared to ViT-L/16, the accuracy of the proposed method without distillation decreases by 3.01%. The

TABLE VIII
ABLATION STUDY OF DIFFERENT LOSS FUNCTIONS IN MODEL ENSEMBLE

Model	GT	$\mathcal{L}_{pred}$	$\mathcal{L}_{token}$	Accuracy
ViT-L/16	✓	×	×	88.77 %
DeViTs w/o supervision	✓	×	×	85.76 %
DeViTs	×	✓	×	86.03 %
	×	×	✓	86.46 %
	×	✓	✓	**87.98 %**

TABLE IX
COMPARISON OF DIFFERENT ENSEMBLE METHODS

Ensemble Methods	Accuracy	Average Latency
Attention	87.09 %	$6.77_{\pm 0.4}$ ms
SENet	88.05 %	$6.17_{\pm 0.6}$ ms
MLP	88.01 %	$6.16_{\pm 0.4}$ ms
Ours	87.98 %	$\mathbf{5.79}_{\pm 0.1}$ ms

Fig. 9.   Results of accuracy and latency for different number of devices. The size of decomposed small models is the same.

results show that distillation can improve the ensemble performance of small models and reduce the accuracy loss caused by decomposition and shrinking. We use two types of loss functions to help the ensemble learning of small decomposed models. Each loss function is helpful, and the best result is achieved when both terms are adopted. The minimal accuracy loss is 0.79% when both two loss functions are adopted.

The comparison results of different ensemble strategies are shown in Table IX. Three competitive ensemble strategies are compared: i) The *Attention* method takes a simple key-value form to aggregate features. ii) The *SENet* [38] computes the importance of different features and weightedly sum these features. iii) The *MLP* approach uses an MLP block to fuse different features. From the table, we observe that SENet achieves the highest accuracy but also the highest inference latency compared to MLP. We remove the activation function in the MLP block and find that it can improve latency by $1.06\times$ with only 0.03% accuracy loss. The slight accuracy loss is negligible compared with the greatly reduced latency. This demonstrates that our method can effectively trade off latency and accuracy.

Effect of Device Number: We analyze the influence on accuracy and latency when changing the number of devices. We employ ViT-L/16 as the decomposed huge ViT and measure the classification accuracy and inference latency in the GPU server varying the number of devices from 2 to 6. The results on CIFAR-100 is shown in Fig. 9. As the number of devices increases, the accuracy is closer to the huge ViT while the latency remains essentially consistent. Our method can achieve $2.7\times$ speed-up with only an approximate 1 % accuracy drop even if there are 6 devices. The result verifies that our method is robust to the number of device number and achieve efficient inference.

VI. RELATED WORKS

A. Efficient ViT

ViTs achieve superior performance by substantially increasing the number of parameters in intelligent applications [39]. We analyze the prior work of ViT training on a multi-GPU server
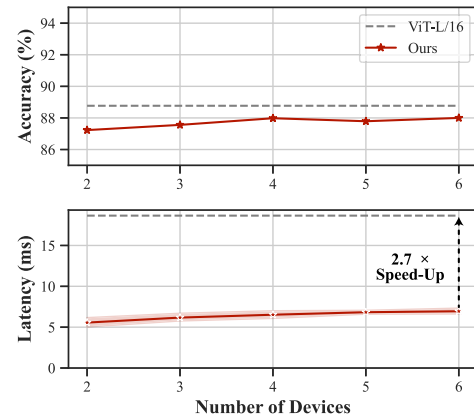
and efficient ViT inference. Then we compare these methods with ours.

ViT Training: Training large-scale ViTs is challenging due to the limitations imposed by GPU memory capacity and substantial computational cost. Most studies for training these transformers within a multi-GPU server aim to reduce GPU memory requirements in a single GPU and communication consumption, hence promoting efficient training. These approaches leverage parallelism strategies and fall into three broad categories: i) *Data parallelism:* Shoeybi et al. [40] proposed to partition the original dataset into small subsets and distribute them across multiple GPUs for training. ii) *Model parallelism:* The huge transformer model is split across and by neural network layers into partitions processed on separate GPUs [41], necessitating communication and transfer of intermediate results for inference or model training. iii) *Pipeline parallelism:* GPipe [42] utilizes pipeline parallelism by dividing consecutive groups of layers into cells positioned on separate GPUs.

ViT Inference: Given the impressive performance of large-scale ViTs, there is an urgent need to compress them for efficient edge inference. Recent studies have proposed lightweight architectures to enhance performance. For instance, Mehta et al. [12] incorporated convolution into transformers, combining the strengths of convolution and attention. Moreover, a series of methods apply traditional model compression techniques to achieve compact ViTs, such as network pruning [43], knowledge distillation [44], [45], [46] and low-bit quantization [47]. Zhu et al. [43] introduced learnable coefficients to evaluate the importance of linear matrix, and neurons with small coefficients are removed. Hao et al. [45] utilized patch-level information to help compact student models imitate teacher models. Yao et al. [47] proposed an efficient and affordable post-training quantization approach to compress large transformers.

In terms of ViT training on a multi-GPU server, the effectiveness of data parallelism is constrained by network bandwidth and the size of model. While model parallelism and pipeline parallelism can reduce the size of the model on a single GPU, the communication overhead remains substantial due to frequent data transmission between different nodes. Regrading to efficient ViT inference, these methods omit the divisibility of ViTs and cannot achieve a satisfactory trade-off between

performance and efficiency. To mitigate these limitations, we propose to decompose a large-scale ViT into multiple smaller models, each tailored based on the significance of their structural components. These compact models are executed in parallel and their intermediate results are transferred to the central node for only a single communication. Moreover, we first develop a novel feature matching module that facilitates the learning of decomposed models from the large-scale ViT, thereby minimizing information loss.

B. Collaborative Inference

Collaborative inference involves partitioning a DNN into two or more segments and deploying them across various devices for inference. This partitioning of DNNs can be executed layer-wise or across layers. We investigate these two types of approaches, layer-wise splitting and across layer splitting, followed by a comparison with our methods.

Layer-Wise Splitting: A DNN is divided into multiple layer groups and distributed across different mobile edge nodes. The front part of the DNN is first executed on the lower-performance edge device, and the obtained intermediate features are offloaded to an edge server or the cloud. Most work in this category focuses on deploying partitioned DNNs into distinct devices based on their computational performance, aiming for an optimal balance between performance and efficiency [9], [48], [49], [50]. For example, Kang et al. [9] demonstrated that judicious selection of a splitting point can decrease latency and energy consumption compared to executing DNNs entirely on the cloud or mobile devices. Tang et al. [50] proposed a deep reinforcement learning algorithm to obtain the optimal task offloading strategy. Another line of work employs quantization [51] or encoding methods [48] to compress intermediate features. Yao et al. [48] utilized an auto-encoder to compress features and then decode them in the cloud. Hao et al. [52] integrated quantization and auto-encoder to alleviate communication overheads. Concurrently, they proposed the MAHPPO algorithm, an innovative solution to the multi-agent collaborative inference problem.

Across Layer Splitting: The DNN weight is generally split into a hierarchy of multiple groups that have no connection among them, allowing parallel execution and model size reduction. These methods require an additional training procedure where publicly available pre-trained models cannot be used after splitting, compared to layer-wise splitting. For instance, Kim et al. [10] split the DNNs into either a set or a hierarchy of multiple groups to create a tree-structured model. These partitioned models can be executed in parallel to accelerate inference. Some methods utilize progressive slicing mechanisms to partition neural models into multiple components to fit in heterogeneous devices. Kim et al. [53] proposed an inference runtime by splitting DNNs across layers. The partitioned models are deployed on the heterogeneous processors of an edge device. Mohammed et al. [11] divided a DNN into multiple partitions that can be processed locally by end devices or offloaded to one or multiple powerful nodes.

While layer groups split by layers are easily deployed on edge devices, layer splits necessitate the transfer of semi-processed input to the subsequent node and the output to the user, thereby increasing the overall execution time. Across layer splitting methods require frequent intercommunication among network splits, considerably increasing the communication overheads. Furthermore, these methods primarily focus on CNNs and cannot be directly applied to the deployment of ViTs. To the best of our knowledge, we first propose a collaborative inference framework for general ViTs in edge devices by decomposing the large ViT into multiple small models.

VII. CONCLUSION

In this paper, we proposed a novel framework to achieve efficient collaborative inference for ViTs, where we first decompose the large ViT into multiple small models and deploy these models on edge devices. Then we designed the DEKD ensemble algorithm to fuse multiple decomposed models and compensate for the accuracy loss caused by decomposition. We also designed a feature matching module to facilitate intermediate knowledge distillation given heterogeneous backbones. We conducted extensive experiments to verify the effectiveness of the proposed framework. The results showed that our framework can dramatically accelerate the inference and reduce the energy consumption on edge devices, which is difficult for existing works. Considering that we only focus on the homogeneous devices in this work, we intend to analyze the effect of device heterogeneity and explore how to achieve efficient collaborative inference on heterogeneous edge devices in the future.

REFERENCES

[1] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. 30: Annu. Conf. Neural Inf. Process. Syst.*, Long Beach, CA, USA, 2017, pp. 5998–6008.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2019, pp. 4171–4186.

[3] Y. Gong, Y. Chung, and J. R. Glass, "AST: Audio spectrogram transformer," in *Proc. 22nd Annu. Conf. Int. Speech Commun. Assoc.*, Brno, Czechia, 2021, pp. 571–575.

[4] Z. Liu et al., "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 10012–10022.

[5] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. 9th Int. Conf. Learn. Representations*, 2021, pp. 1–21.

[6] M. Masoudi and C. Cavdar, "Device vs edge computing for mobile services: Delay-aware decision making to minimize power consumption," *IEEE Trans. Mobile Comput.*, vol. 20, no. 12, pp. 3324–3337, Dec. 2021.

[7] J.S. Jo Jeong and P. Kang, "Benchmarking GPU-accelerated edge devices," in *Proc. IEEE Int. Conf. Big Data Smart Comput.*, 2020, pp. 117–120.

[8] A. A. Süzen, B. Duman, and B. Şen, "Benchmark analysis of Jetson tx2, Jetson nano and Raspberry pi using deep-CNN," in *Proc. Int. Congr. Human-Comput. Interact.*, 2020, pp. 1–5.

[9] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2017, pp. 615–629.

[10] J. Kim, Y. Park, G. Kim, and S. J. Hwang, "SplitNet: Learning to semantically split deep networks for parameter reduction and model parallelization," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 1866–1874.

[11] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *Proc. IEEE 39th Conf. Comput. Commun.*, 2020, pp. 854–863.

[12] S. Mehta and M. Rastegari, "MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer," in *Proc. 10th Int. Conf. Learn. Representations*, 2022, pp. 1–26.

[13] M. Chen, H. Peng, J. Fu, and H. Ling, "AutoFormer: Searching transformers for visual recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Montreal, QC, Canada, Oct. 10–17, 2021, pp. 12250–12260.

[14] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 10347–10357.

[15] NVIDIA, "Nvidia Jetson modules," 2022. [Online]. Available: https://www.nvidia.cn/autonomous-machines/embedded-systems/jetson-xavier-nx/

[16] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does BERT look at? an analysis of BERT's attention," in *Proc. ACL Workshop BlackboxNLP: Analyzing Interpreting Neural Netw. NLP*, Florence, Italy, 2019, pp. 276–286.

[17] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?," in *Proc. Adv. Neural Inf. Process. Syst. 32: Annu. Conf. Neural Inf. Process. Syst.*, 2019, pp. 14014–14024.

[18] A. Khakzar, S. Baselizadeh, S. Khanduja, C. Rupprecht, S. T. Kim, and N. Navab, "Neural response interpretation through the lens of critical pathways," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 13528–13538.

[19] Q. Guo et al., "Online knowledge distillation via collaborative learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Seattle, WA, USA, Jun. 13–19, 2020, pp. 11017–11026.

[20] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, "Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned," in *Proc. 57th Conf. Assoc. Comput. Linguistics*, 2019, pp. 5797–5808.

[21] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, "DynaBERT: Dynamic BERT with adaptive width and depth," in *Proc. Adv. Neural Inf. Process. Syst. 33: Annu. Conf. Neural Inf. Process. Syst.*, 2020, pp. 9782–9793.

[22] X. Jiao et al., "TinyBERT: Distilling BERT for natural language understanding," in *Proc. Findings Assoc. Comput. Linguistics: EMNLP*, 2020, pp. 4163–4174.

[23] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "MobileBERT: A compact task-agnostic BERT for resource-limited devices," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 2158–2170.

[24] H. Wang, S. Bao, L. Huang Dong, and F. Wei, "MiniLMv2: Multi-head self-attention relation distillation for compressing pretrained transformers," in *Proc. Findings Assoc. Comput. Linguistics: ACL/IJCNLP*, 2021, pp. 2140–2151.

[25] NVIDIA, "Nvidia jetson nano," 2022. [Online]. Available: https://www.nvidia.cn/autonomous-machines/embedded-systems/jetson-nano/

[26] T. L. Foundation, "Tc-show / manipulate traffic control settings," 2022. [Online]. Available: https://www.linux.com/tutorials/tc-show-manipulate-traffic-control-settings/

[27] Google, "gprc - a rpc library and framework," 2022. [Online]. Available: https://grpc.io/

[28] R. Wightman, "Pytorch image models," 2022. [Online]. Available: https://github.com/rwightman/pytorch-image-models

[29] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," M.S. thesis, Univ. Toronto, Toronto, ON, Canada, 2009.

[30] M. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Proc. IEEE 6th Indian Conf. Comput. Vis.*, 2008, pp. 722–729.

[31] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3D object representations for fine-grained categorization," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, 2013, pp. 554–561.

[32] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[33] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi, "Escaping the Big Data paradigm with compact transformers," 2021, *arXiv:2104.05704*.

[34] L. Yuan et al., "Tokens-to-token ViT: Training vision transformers from scratch on ImageNet," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 538–547.

[35] X. Chu et al., "Twins: Revisiting the design of spatial attention in vision transformers," in *Proc. Adv. Neural Inf. Process. Syst. 34: Annu. Conf. Neural Inf. Process. Syst.*, 2021, pp. 9355–9366.

[36] PyTorch, "Pytorch profiler," 2022. [Online]. Available: https://pytorch.org/docs/stable/profiler.html

[37] M. S. Inc., "High voltage power monitor," 2022. [Online]. Available: https://www.msoon.com/high-voltage-power-monitor

[38] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 2011–2023, Aug. 2020.

[39] D. Narayanan et al., "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2021, Art. no. 58.

[40] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv: 1909.08053*.

[41] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "FireCaffe: Near-linear acceleration of deep neural network training on compute clusters," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2592–2600.

[42] Y. Huang et al., "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst. 32: Annu. Conf. Neural Inf. Process. Syst.*, 2019, pp. 103–112.

[43] M. Zhu, Y. Tang, and K. Han, "Vision transformer pruning," 2021, *arXiv:2104.08500*.

[44] Z. Hao, Y. Luo, Z. Wang, H. Hu, and J. An, "Model compression via collaborative data-free knowledge distillation for edge intelligence," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2021, pp. 1–6.

[45] Z. Hao et al., "Learning efficient vision transformers via fine-grained manifold distillation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 9164–9175.

[46] Z. Hao, Y. Luo, Z. Wang, H. Hu, and J. An, "CDFKD-MFS: Collaborative data-free knowledge distillation via multi-level feature sharing," *IEEE Trans. Multimedia*, vol. 24, pp. 4262–4274, 2022.

[47] Z. Yao, R. Y. Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 27168–27183.

[48] S. Yao et al., "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proc. 18th ACM Conf. Embedded Netw. Sensor Syst.*, 2020, pp. 476–488.

[49] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[50] 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 Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 4039–4056, Dec. 2022.

[51] 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.*, 2018, pp. 671–678.

[52] Z. Hao, G. Xu, Y. Luo, H. Hu, J. An, and S. Mao, "Multi-agent collaborative inference via DNN decoupling: Intermediate feature compression and edge learning," *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 6041–6055, Oct. 2022.

[53] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "layer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in *Proc. 14th EuroSys Conf.*, 2019, pp. 45:1–45:15.

Guanyu Xu received the BE degree from the Beijing Institute of Technology, China, in 2021. He is currently working toward the PhD degree with School of Information and Electronics, Beijing Institute of Technology, China. His research interests include model compression and edge intelligence.

Zhiwei Hao received the BS degree in applied physics from the Beijing Institute of Technology, China, in 2019. He is currently working toward the PhD degree with the School of Information and Electronics, Beijing Institute of Technology, China. His research interests include deep learning and edge computing.

Yong Luo (Member, IEEE) received the BE degree in computer science from the Northwestern Polytechnical University, Xi'an, China, and the DSc degree in the School of Electronics Engineering and Computer Science, Peking University, Beijing, China. He is currently a professor with the School of Computer Science, Wuhan University, China. His research interests are primarily on machine learning and data mining with applications to visual information understanding and analysis. He has authored or co-authored more than 50 papers in top journals and prestigious conferences including *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Transactions on Neural Networks and Learning Systems*, *IEEE Transactions on Image Processing*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Multimedia*, WWW, IJCAI, AAAI, CIKM, ICDM and ICME. He is serving on editorial board for IEEE T-MM. He received the IEEE Globecom 2016 Best Paper Award, and was nominated as the IJCAI 2017 Distinguished Best Paper Award. He is also a co-recipient of the IEEE ICME 2019 and IEEE VCIP 2019 Best Paper Awards.

Han Hu (Member, IEEE) received the BE and PhD degrees from the University of Science and Technology of China, Hefei, China, in 2007 and 2012, respectively. He is currently a professor with the School of Information and Electronics, Beijing Institute of Technology, Beijing, China. His research interests include multimedia networking, edge intelligence, and space-air-ground integrated network. Dr. Hu received several academic awards, including the Best Paper Award of IEEE TMM, in 2023, the Best Paper Award of IEEE TCSVT in 2019, the Best Paper Award of IEEE Multimedia Magazine, in 2015, and the Best Paper Award of IEEE Globecom in 2013. He served as an associate editor of IEEE Transactions on Multimedia and Ad Hoc Networks, and a TPC Member of Infocom, ACM MM, AAAI, and IJCAI.

Jianping An (Member, IEEE) received the PhD degree from Beijing Institute of Technology, Beijing, China, in 1996. He is currently a full professor and the dean of the School of Cyberspace Science and Technology, Beijing Institute of Technology. His research interests are in the field of digital signal processing, cognitive radio, wireless networks, and high-dynamic broadband wireless transmission technology.

Shiwen Mao (Fellow, IEEE) received the PhD degree in electrical engineering from Polytechnic University, in 2004. He is a professor and Earle C. Williams Eminent Scholar, and director of the Wireless Engineering Research and Education Center with Auburn University. He is research interest includes wireless networks, multimedia communications, and smart grid. He received the IEEE ComSoc TC-CSR Distinguished Technical Achievement Award, in 2019 and the NSF CAREER Award, in 2010, and several service awards from IEEE ComSoc. He is a co-recipient of the 2021 Best Paper Award of Elsevier/KeAi Digital Communications and Networks Journal, the 2021 IEEE Internet of Things Journal Best Paper Award, the 2021 IEEE Communications Society Outstanding Paper Award, the IEEE Vehicular Technology Society 2020 Jack Neubauer Memorial Award, the 2018 Best Journal Paper Award and the 2017 Best Conference Paper Award from IEEE ComSoc MMTC, and the 2004 IEEE Communications Society Leonard G. Abraham Prize in the Field of Communications Systems. He is a co-recipient of the Best Paper Awards from IEEE ICC 2022 and 2013, IEEE GLOBECOM 2019, 2016, and 2015, and IEEE WCNC 2015, and the Best Demo Awards from IEEE INFOCOM 2022 and IEEE SECON 2017.