MDP: Model Decomposition and Parallelization of Vision Transformer for Distributed Edge Inference

Weiyan Wang[1], Yiming Zhang[2], Yilun Jin[1], Han Tian[1], Li Chen[3]
[1]Hong Kong University of Science and Technology, [2] Xiamen University, [3] Zhongguancun Lab.
wwangbc@connect.ust.hk, sdiris@gmail.com, {yilun.jin,htianab}@connect.ust.hk, crischenli@gmail.com

***Abstract*—Distributed edge inference emerges to be a promising paradigm to speed up inference. Previous works make physical partitions on CNNs to realize it, but there are the following challenges for vision transformers: (1) high communication costs for the large model; (2) stragglers because of heterogeneous devices; (3) time-out exceptions due to unstable edge devices.**

Therefore, we propose a novel Model Decomposition and Parallelization(MDP) for large vision transformers. Inspired by the implicit boosting ensemble in the vision transformer, MDP decomposes it into an explicit boosting ensemble of different and parallel sub-models. It sequentially trains all sub-models to gradually reduce the residual errors. To minimize dependency and communication among sub-models, We adopt stacking distillation to bring every sub-model extra information about others for better error correction. Different sub-models can take both different image sizes and model sizes to run on heterogeneous devices and improve the ensemble diversities. To handle the time-out exception, we add vanilla supervised learning on every sub-model for the bagging ensemble in case of the early termination of boosting ensemble. As a result, all sub-models can not only run in parallel without much communication but also can be adapted to the heterogeneous devices, while maintaining accuracy even with time-out exceptions. Experiments show that MDP can outperform other baselines by $5.2\times \sim 2.1\times$ in latency and $5.1\times \sim 1.7\times$ in throughput with comparable accuracy.

***Index Terms*—Distributed edge inference, vision transformers, boosting ensemble,**

I. Introduction

Recently, the distributed edge inference emerges to be a promising paradigm to speed up inference with multiple edge devices [1], [2], [3], [4], especially in remote scenarios like security surveillance in the ocean liner, fire detection in the primary forest, and animal tracking in the wild desert. There are two reasons for its rising [5], [6]. One is that inference tasks prefer directly running on the edge due to the high transferring overhead or unavailability of the Internet. The other is that there are massive local edge devices(e.g., IoT sensors and embedded controllers) to offer abundant computation resources. Some previous works [1], [2], [3], [4] have explored the proper physical partition on traditional CNNs so that inference can efficiently run on multiple heterogeneous edge devices connected to the local network(§II-B2).

Inspired by their success in Natural Language Processing (NLP), large transformer models like ViT [7] and MAE [8] have been widely adopted by Computer Vision (CV) (§II-A). After being pre-trained on massive images, vision transformers can be further fine-tuned on the down-streaming tasks for superior accuracy compared to traditional CNNs [7], [8].

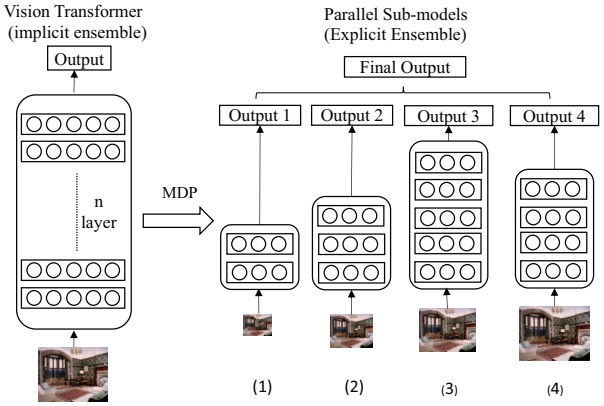


Figure 1: MDP decomposes the vision transformer into a group of sub-models: (1) decoupled and parallel sub-models without much dependency and communication; (2) different sub-models can have different image sizes and layer numbers for heterogeneous devices.

Despite the high accuracy, the vision transformer has high inference latency on the edge device. Compared with traditional CNNs, vision transformer consumes higher computation and larger storage than convolutions due to the complex self-attention mechanism. Unlike local and spatial convolutions, self-attention has a global dependency on the entire image. And all attention layers keep the same large feature size of the original image and high hidden dimensions like 768. Some works have explored to reduce its inference cost, including quantization [9], [10], knowledge distillation [11], [12], model pruning [13], [14], and patch slimming [15]. But they all only generate a single model for one edge device, which cannot save over 50% inference latency to maintain the accuracy(§II-B1). Moreover, because all these models compressed from the same original model have low diversities, their direct bagging ensemble has limited improvement.

Furthermore, it has the following problems with directly applying physical partitions on the vision transformer for efficient distributed edge inference:

- **High communication cost:** Although local networks like LAN or Wi-fi are faster than the Internet, it is much slower than the data center network. Therefore, the vision transformer has a high overhead to transfer its large intermediate results.
- **Straggler problem:** The edge devices inherently have heterogeneities in computation and storage. But attention

979-8-3503-5826-1/23/$31.00 ©2023 IEEE
DOI 10.1109/MSN60784.2023.00086

layers are hard to be properly partitioned due to their global dependency, resulting in straggler problems.

- **Time-out exception:** Unlike stable servers in the data center, edge devices are not exclusive and are always available for distributed edge inference. This time-out problem is ignored by all previous works.

To this end, we propose the novel Model Decomposition and Parallelization(MDP) of the vision transformer for efficient distributed edge inference. Inspired by the implicit boosting ensemble in the vision transformer (§II-A), MDP decomposes the original model into an explicit boosting ensemble of different sub-models running in parallel, instead of the physical model partition(§IV). To achieve this, all sub-models are sequentially trained to gradually reduces the error between the ground truth and the existing sub-models as more sub-models added. To minimize dependency and communication among sub-models, we bring up the virtual stacking that every sub-model distills all previous sub-models into its intermediate layers. Similar to the stacked attention layers in the vision transformer, every sub-model can better reduce the error by estimating the output of all the previous sub-models. Therefore, all sub-models can work together to maintain accuracy with little dependency and communication during inference.

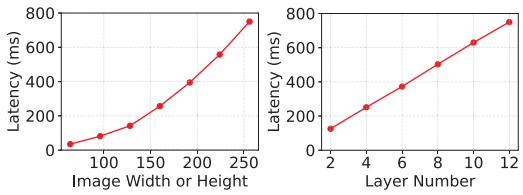
Additionally, all sub-models can have not only various image sizes but also different model sizes to run on heterogeneous edge devices. We feed all sub-models with different scales of images to adjust the workloads among devices and build up the feature pyramid. Moreover, we prune all sub-models to have different layer numbers for different devices, which can also improve ensemble diversities from the different network architectures of all sub-models.

Finally, we also employ the extra bagging ensemble to handle the time-out exception (§V-B). In boosting ensemble, if the m-th sub-model fails to return the output in time, all its following sub-models cannot participate the boosting ensemble. Therefore, we add an extra output classifier for vanilla supervised learning on every sub-model. Then the sub-models following the failed one can contribute to the bagging ensemble in case of wasting their computation.

We have implemented a prototype of the proposed MDP for the vision transformer and conduct comprehensive experiments to verify its efficiency and effectiveness. Experiment results(§VI) show that it improves the inference latency over all baselines by $5.2\times \sim 2.1\times$, outperforms the others by $5.1\times \sim 1.7\times$ in terms of throughput, and achieve almost the same accuracy as the original vision transformer.

In summary, our work makes the following contributions:

1) We propose MDP to decompose the original vision transformer into a group of virtually stacked sub-models to run in parallel with little communication.
2) We adjust both input image and model sizes in different sub-models to be better adapted to the heterogeneous devices, meanwhile improving the ensemble diversity.
3) We handle the time-out exception by extra bagging ensemble, which is a problem ignored by all previous works.



(a) Image Size (b) Model Size

Figure 2: Different factors influencing the inference latency

II. Background and Motivation

A. Vision Transformer

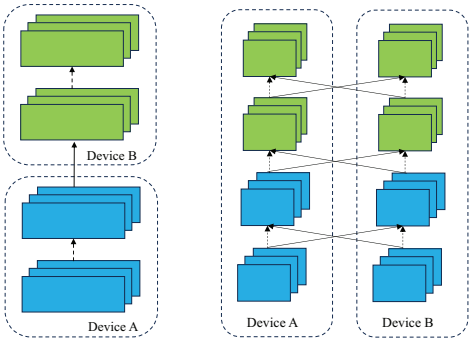
Transformer models have grown to be the universal model architecture, because of their unprecedented achievements in NLP, CV, and multi-modality. By following the learning paradigm of BERT [16] and GPT [17], vision transformers like ViT [7] and MAE [8] are firstly pre-trained on large unlabeled data and then fine-tuned on the small labeled dataset. Because of their effective pre-training, vision transformers can significantly outperform previous works like ResNet [18].

Initially, the vision transformer splits one image into multiple patches as the "sequence tokens" in CV [7]. All patches are non-overlapping and in the same size, so the patch number is determined by the image size. Then it conducts a linear projection on every patch for the patch embeddings, which forms the "token" sequence fed to the attention layers.

To capture global semantic relationships among patches, every attention layer employs a complex self-attention mechanism. It computes the dot-product similarities between all patch pairs to generate an attention weight matrix, which has $O(N^2)$ time complexities. Every position generates the new patch feature by aggregating all attention-weighted patch features. Then it applies a non-linear transformation on the patch features separately and identically.

All the layers are stacked with residual connections, which can be interpreted as the implicit boosting ensemble of all layers [19], [20]. The residual connection adds the original input of the $i-th$ layer x_i back to the result of its layer function $F_i(x_i)$, so its layer output is $H_i = F_i(x_i) + x_i$, namely the refinement of its input. Because there are numerous attention layers stacked with residual connections. Then the output of the n-th layer is expanded as $H_n = F_n(\sum_{i=1}^{n-1} H_i + x_0) + \sum_{i=1}^{n-1} H_i + x_0$. Thus, residual connections enable the multi-path forward in the vision transformer, which behaves like the ensemble of relatively shallow networks [19], [20].

Compared with traditional CNNs, the vision transformer has three main differences. First, self-attention has a global dependency on the entire image, but convolution only computes local spatial relationships of neighboring pixels. Furthermore, all attention layers keep the same patch number(i.e., the same image scale) to be compatible with residual connections. But there are some poolings or strides inserted into CNNs to subsample the image scale in different layers [18], [21] Finally, all attention layers also have the same hidden dimensions like 768, different from the increasing convolution channels from the bottom to the top layers.



(a) Horizontal (b) Vertical

Figure 3: Different Workload Partitions

Despite the high accuracy, vision transformers have high computation costs. As CPU inference results in Figure 2, there are the following two factors affecting the inference latency:

- **Image Size:** inference latency significantly increases with the image size(or patch number N) as shown in Figure 2a, because of $O(N^2)$ time complexity in attention mechanism [22].
- **Model Size:** vision transformer relies on large model depth L to extract effective semantic features, but it linearly increases time complexity as $O(L)$ [23] as shown in Figure 2b.

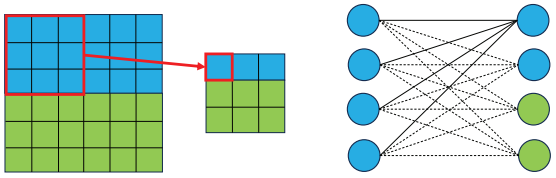
B. Limitations of Existing Works

There are two categories of previous works about efficient inference on edge devices. One is the model compression of vision transformers for the single edge device. The other is workload physical partition for distributed edge inference.

1) Model Compression for the Edge Device: Some existing works have explored compressing the large vision transformer to run on the edge device, but they all overlook the possibility of exploiting multiple edge devices for distributed inference. These works only focus on generating the single small model equivalent to the large one, which also has limited room to be further compressed for less computation costs.

There are various works for the model compression of vision transformers. Quantization [9] studies how to replace expensive float parameters with cheap low-bit quantized parameters, but it cannot reduce the time and space complexities. Knowledge distillation [11] trains the small student model to mimic the behavior of the teacher vision transformer, which needs a large enough student model. Model pruning [13] recognizes and prunes redundant weights, neurons, and attention heads to save computation costs, but most model parts still remain to maintain accuracy, and the sparsely remained model is not friendly to efficient hardware execution. Patch slimming [15] employs extra modules to recognize and discard the redundant patches in every layer, but it has extra computations and the initial patch number is fixed.

2) Physical Partition for Distributed Edge Inference: Previous works make physical partitions on CNNs to distribute proper workloads across different edge devices. Although every edge device has limited resources, multiple edge devices



(a) Local dependency (b) Global dependency

Figure 4: Dependency Comparison between the convolution layer(a) and the fully-connected layer(b)

can cooperate to speed up the model inference. As shown in Figure 3, there are two kinds of model partitions, including horizontal partition and vertical partition.

Horizontal partition borrows the idea of naive model parallelism [24], [25] in the distributed training. It splits the deep model into different layer parts to run on different devices. As illustrated in Figure 3a, the input data first run through the bottom layers on device A, and then the intermediate features will be transferred to device B for the following layer computation. If intermediate features are large, the communication costs can be high. Additionally, the inference on the single sample is sequentially executed in a pipeline on multiple devices. Therefore, the horizontal partition can not improve the inference latency for individual samples, and it needs enough data to fulfill the pipeline.

As shown in Figure 3b, the vertical partition explores parallel computation within the layer for distributed edge inference. Some works [26], [27], [2] observe and exploit the local dependency on the input of some layers. For example, the convolution layer assumes that CV data have local spatial correlations, so it only takes a small convolution window(e.g., 3x3) sliding over the entire image. As shown in Figure 4a, every output value only depends on the neighboring input values in the window, instead of all inputs. Similarly, there are some other locally dependent layers, such as pooling, batch normalization, and element-wise activations like RELU. Therefore, these works [26], [27], [2] split the input features maps into multiple tiles to be processed in parallel among different edge devices. However, the partition on input features is not feasible for the vision transformer, because of its global dependency on the entire image.

For the global dependent layers like fully-connected and attention layers, tensor parallelism [28], [29] can partition the layer itself into different parallel parts to generate different outputs as shown in Figure 4b. For example, the weight matrix of fully-connected layers can be split into multiple chunks and the attention layer inherently has multiple parallel heads. Since every device only has partial results, it needs extra all-gather communications to concatenate the whole results for the following layer. In the distributed training on servers, the tensor parallelism usually happens only among the GPUs in the same node to take advantage of the fast PCI-E transferring. However, the distributed edge inference only has the local network much slower than the PCI-E and data center network.

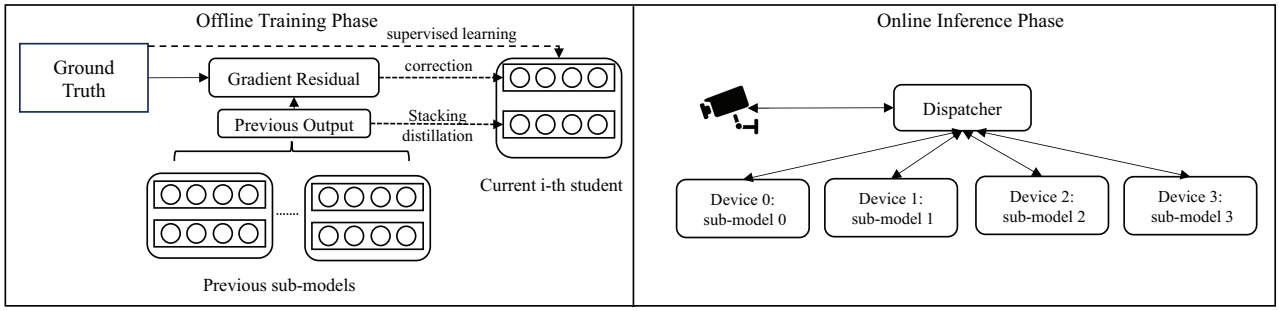


Figure 5: Overview of the two phases in MDP : (1) offline training phase: sequentially training different sub-models with boosting ensemble loss, virtual stacking loss, and auxiliary supervised learning loss ; (2) online inference phase: one device works as the master to coordinate the distributed inference, while all devices execute different sub-models respectively.

III. Framework Overview

As the overview shown in Figure 5, MDP have two different phases, namely offline training and online inference. The offline training is the first phase that generates different sub-models(§IV). And the second one is online inference(§V), which exploits multiple edge devices for the distributed inference of different sub-models. Specifically, the two different phases play the following roles:

- **Offline Training:** It sequentially trains all sub-models in explicit boosting ensemble to gradually reduce the error. To reduce the dependency and communication during the inference, MDP employs virtual stacking to bring in extra information about previous sub-models. Additionally, different sub-models can have different input and model sizes to run on heterogeneous devices, which also improves ensemble diversities. It also employs supervised learning as the auxiliary task to handle time-out exceptions.
- **Online Inference:** Unlike training requiring the outputs of previous sub-models as the labels, all sub-models are free of labels to run in parallel during the inference. MDP distributed inference data across different devices, and all sub-models are combined for the final output. It also considers how to handle some unexpected time-out events for the distributed edge inference, in case any sub-model output is missing.

IV. Offline Training

In this section, we first introduce the overall workflow of offline training in MDP . Then, we describe the all training loss terms. Finally, we show how different sub-models are adapted to heterogeneous edge devices.

A. Overall Workflow

At its heart, MDP decomposes the large vision transformer into an equivalent group of different and parallel sub-models for future distributed edge inference. The process of how MDP generates the group of sub-models is described in the Algorithm 1. And there are the following key steps:

1) **Device profiling** (line: 2): In the initialization, MDP firstly profile the capabilities of local heterogeneous edge devices. Then it can assign different sub-models to different edge devices for suitable computation and storage costs.
2) **Image pre-processing** (line: 10): According to the heterogeneous capabilities, different sub-models can take images resized into different scales as their inputs. Meanwhile, MDP can build the feature pyramid of multiple scales [21] from all sub-models for better prediction. Moreover, it applies Mix-up [30] on training data as the data augmentation. And it sub-samples the whole datasets to get the subsets of samples having large gradient residuals.
3) **Dynamic layer pruning** (line: 13-18 and 20): Besides the image size, MDP can further adapt the model size to the edge devices by dynamic layer pruning. During the training of every data batch, it dynamically skips some layers with every-other strategy [23]. After completing training one sub-model, it can prune the model layers for better adaptation to the assigned device.
4) **Boosting ensemble** (line: 14): All sub-models are sequentially trained to work together in the form of boosting ensemble. Every sub-model is trained to reduce the error between ground truth and all previous sub-models. As a result, MDP can gradually improve the accuracy as new sub-models are added.
5) **Virtual stacking** (line: 15): To eliminate dependency and reduce communication, we bring in the novel virtual stacking loss to train sub-models. The virtual stacking distillates all the previous sub-models into the intermediate layer of the current one. Therefore, every sub-model is free of physical dependency on others, but it reserves the others' information within itself for better error correction.
6) **Supervised learning** (line: 16): All sub-models are also trained on the ground truth labels with vanilla the supervised learning loss. Therefore, if the output of any sub-model is missing, the following sub-models can also contribute to the bagging ensemble (§ V-B).

B. Sub-Model Training Loss

To gradually reduce the error, MDP continue to add a new sub-model until it is overfitting or exceeds the device number. Every newly added sub-model is trained to further reduce the residual error between the ground truth and all previous sub-models. Except that the first sub-model directly trains on the ground truth, other sub-models learn to correct the error of their previous sub-models. Formally, the final output loss of

Algorithm 1 the Process of MDP

Require: pretrainVT, devices
#Profiling all device capabilities
imgSizes = deviceProfiling(devices)
#Sequential training of all sub-models
subModels = []
while not overfitting or len(subModels) $<$ deviceNum **do**
  newSubmodel = PretrainVT()
  #Training of current sub-model
  while training **do**
    #Image Pre-processing
    x, y = mixup(x, y, imgSizes[i])
    outputs = newSubmodel(x)
    # Dynamical layer dropping
    for layerGap $\leftarrow$ 12 to 2 **do**
      Compute the boosting loss as Eqn 1
      Compute the virtual stacking loss as Eqn 2
      Compute the supervised learning loss as Eqn 3
      Accumulate the gradients of current layers
    end for
    Update newSubmodel with all the gradients
  end while
  # Adaptive layer pruning for current sub-model
  Prune newSubmodel for the proper layer number
  subModels.append(newSubmodel)
end while

the i-th sub-model $L_1^{(i)}$ is defined as:

$$L_1^{(i)}(x) = \begin{cases} SCE(S^{(i)}(x), y), & i = 0 \\ SCE(S^{(0)}(x) + \sum_{j=1}^{i} \alpha_j S^{(j)}(x), y) & i > 0. \end{cases} \quad (1)$$

In the equation, SCE stands for the soft target cross-entropy loss function. The y is the ground-truth label of data x, which is from mix-up data augmentation. $S^{(i)}(x)$ denotes the final output of the i-th sub-model. α_j is the shrinkage rate for the gradient boosting, which is also trainable. $S^{(0)}(x) + \sum_{j=1}^{i} \alpha_j S^{(j)}(x)$ is the boosting ensemble of the previous i models. Only $S^{(i)}$ and α_i is trainable, while the other models $S^{(j<i)}$ and $\alpha_{j<i}$ are fixed.

For efficient distributed edge inference, it prefers all sub-models to be decoupled from each other to run independently without dependency and communications. However, the original vision transformer sequentially stacks attention blocks to continuously refine the feature for better prediction. Similarly, we bring in the novel virtual stacking for the parallel sub-models, which reserves the information of other sub-models in the boosting ensemble.

Specifically, every sub-model distills all the previous sub-models into its intermediate layer to realize the virtual stacking of sub-models. Except for the first one, every sub-model adds another classifier on its intermediate layer in the half depth, so it can distill the ensemble of all previous sub-models into its bottom parts. The training loss $L_1^{(i)}$ of virtual stacking for the i-th sub-model is shown in the following equation:

$$L_2^{(i)}(x) = SCE(M^{(i)}(x), S^{(0)}(x) + \sum_{j=1}^{i-1} \alpha_j S^{(j)}(x)) \quad (2)$$

where $M^{(i)}(x)$ denotes the output of the additional classifier applied on the middle layer in the i-th sub-model. And only $M^{(i)}$ is trainable in $L_2^{(i)}$, while all the previous sub-models $S^{(j)}(x)$ are fixed.

Furthermore, all sub-models are also trained to make direct predictions independently. This training loss is mainly used to handle unexpected sub-model time-out during inference. Additionally, it also works as multitask learning to improve the representation learning for the boosting ensemble,

$$L_3^{(i)}(x) = SCE(S_{sup}^{(i)}(x), y) \quad (3)$$

Therefore, the overall training loss L for the i-th sub-model can be written as follows:

$$L = L_1 + \lambda_1 L_2 + \lambda_2 L_3 \quad (4)$$

where λ_1 and λ_2 are the hyper-parameters to balance layer distillation and final residual error correction. And the total sub-model number is the total number of edge devices.

C. Sub-Model Adaptation to Heterogeneity

To be adapted to heterogeneous edge devices, all sub-models are designed to have different input sizes and model sizes. MDP employs a greedy strategy to assign the smaller model to process smaller images on weak devices, while powerful devices have a larger model to handle larger images. Therefore, different edge devices can have suitable workloads to reduce the straggler problem caused by heterogeneity.

Specifically, MDP first profiles and sorts out the capabilities of all edge devices. We set the smallest size as 64×64, the original image size as 256×256, and the increasing step size as same as patch size 16×16. Then we start with on the weakest device and make the image size as 64×64 for its sub-model, which is split into 16×16 non-overlapping patches. Then if given a device that has r times larger computation capability than the weakest one, we choose the not used size s that makes $s^2/64^2$ the closest one to r for the device. Besides adjusting the workload to different edge devices, the various sizes of images can also explicitly build up the feature pyramid of different scales for better prediction.

Since image size cannot perfectly fit the device performance differences, all sub-models can dynamically skip some layers to further adjust the workload. For example, if we have two same devices, one can have smaller images but more layers, while the other has larger images but fewer layers. When training one data batch, we dynamically drop different rates of layers by following the every-other strategy [23]. In other words, if the pruning rate is p, it drops the layer at the depth d such that $d \mod \lfloor \frac{1}{p} \rfloor = 0$. Following previous NAS works [31], MDP updates the model parameters with all accumulated gradients of different pruning rates, so the sub-model is robust to different layer numbers. After finishing training one sub-model, it will choose the layer number improving the most accuracy but not having longer latency than any previous one. It can further fine-tune the pruned sub-model to improve

accuracy. And sub-models with different layer numbers can improve the diversity of the final ensemble.

If the device number is larger than the sub-model number, we can replicate or build multiple sub-model groups to run concurrently on different devices for different samples.

D. Training Cost Discussion

Compared with other model compression, MDP only has about 1.5x training time to train all sub-models. We use pre-trained models to initialize our sub-models. Their training data is also the sub-set of the target dataset, which has smaller image sizes. MDP can also use the powerful servers to conduct distributed training for less time costs. Furthermore, the training cost can be amortized during the long-term deployment.

V. Online Inference

In this section, we first introduce the detailed inference procedure. Then we discuss how we make the final prediction from the outputs of all sub-models, and we will introduce how we can handle the time-out exception.

A. Online Inference Procedure

In general, MDP employs the master-worker architecture for the distributed edge inference. The master is the gate and coordinator for the distributed inference, which co-locates with the largest sub-model on the most powerful device. All available edge devices play the role of workers to execute different sub-models in parallel. And the master and all workers are connected to the local network like LAN or Wi-Fi. Specifically, MDP has the following procedures:

1) The camera or user sends the inference data to the master.
2) The master first resizes the image into workers' sizes and then dispatches the resized images to different workers respectively. And it can overlap the resizing computation and data transferring to save time cost.
3) All workers receive the image data and conduct the inference in parallel.
4) The master gathers the outputs from all workers, including the output for boosting ensemble and vanilla supervised learning output.
5) The master sum up the outputs of all workers and return the final prediction to the user.

B. Final Prediction of Sub-model Ensemble

All sub-models are trained in boosting ensemble, so all sub-models can predict by summing up their weighted outputs. If we have n sub-models, the prediction of boosting ensemble is $S^{(0)}(x) + \sum_{j=1}^{N} \alpha_j S^{(j)}(x)$. Additionally, all sub-models also make independent predictions. Therefore, we can exploit the bagging ensemble of their independent predictions and the boosting ensemble to enhance the prediction as follows:

$$B(x) = \frac{1}{n+1}(S^{(0)}(x) + \sum_{j=1}^{N} \alpha_j S^{(j)}(x) + \sum_{j=1}^{N} S_{sup}^{(j)}(x)) \quad (5)$$

Unlike stable servers, there is no guarantee that all edge devices are always exclusive or available for distributed inference. Occasionally, any edge device can be occupied by other tasks. The network bandwidth can be also shared by some other background flows. It can be even unexpectedly turned off due to the exhausted battery. Therefore, time-out exceptions can happen on any edge device. If the outputs of the $m-th$ sub-model do not arrive in time, MDP can still make the final prediction as follows:

$$B(x) = \frac{1}{n}(S^{(0)}(x) + \sum_{j=1}^{m} \alpha_j S^{(j)}(x) + \sum_{j \neq m} S_{sup}^{(j)}(x)) \quad (6)$$

In this equation, we only conduct boosting ensemble of the sub-models before the missing one. To improve the accuracy and avoid the wasting of sub-models after the m-th, we add independent prediction of all the others in the bagging ensemble as written in the last term.

VI. Evaluation

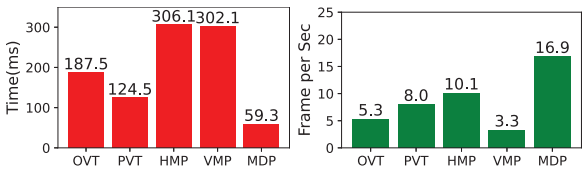
In this section, we report the evaluation results of our MDP to verify its efficiency and effectiveness. We first describe the experiment settings. Then, we show our MDP can outperform others in terms of latency and throughput while maintaining accuracy. Finally, we make some deep-diving experiments.

A. Experiment Settings

1) Implementation: We implement the MDP with about 2k lines of Python code. Specifically, we use the PyTorch [32] as our DL framework, timm from HuggingFace [33] for the vision transformer implementations, and gloo [34] for communications among devices. And we set the minimum image size as 64×64, the original image size as 256×256, and conduct grid searching for both the λ_1 and λ_2 during validation. We initialize our model with the pre-trained MAE [8] and keep all other hyperparameters the same as it, including fixed 16×16 patch size, learning rate schedule, batch size, and so on.

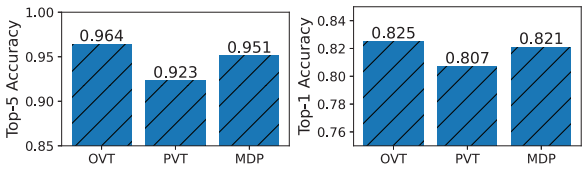
2) Test-bed: Following the previous work [2], [3], [4], we deploy MDP on 4 embedded devices with ARM CPUs @ 2.8 GHz. If the number of available edge devices is larger than the sub-model number, we can simply replicate the sub-model group to run on different devices for different inference samples. All edge devices have PyTorch built from the source for ARM, but they have different hardware configurations. Their CPU core numbers are one, two, three, and four respectively. And their host memory sizes are 2 GB, 4 GB, 8 GB, and 16 GB respectively. All the devices are connected by the LAN with up to 1 Gbps bandwidth.

3) Dataset: We use the famous ImageNet [35] and Indoor Scene Recognition [36] as the datasets for evaluation. There are 1000 image categories in ImageNet, and its total number of images is about 14 million. Indoor Scene Recognition has 15620 images in 67 categories, and there are at least 100 images for each category. We follow the official way to split the datasets for training and testing. We set the top-5 accuracy and top-1 accuracy as the accuracy metrics for ImageNet and Indoor Scene Recognition respectively. Additionally, we set



(a) Latency (b) Throughput

Figure 6: Average latency(the lower is better) and throughput(the higher is better) measurements and comparisons



(a) ImageNet(top-5) (b) Indoor(top-1)

Figure 7: Accuracy comparisons (the higher is better)

the target accuracy for the model compression methods as a 95% accuracy score of the original model.

4) Baselines: We consider the following three baselines further fine-tuned on ImageNet for classification to compare with our proposed MDP : (1) **Original Vision Transformer (OVT)**: we run the MAE pre-trained version of the vision transformer on the most powerful device; (2) **Pruned Vision Transformer (PVT)**: we run the smallest pruned vision transformer that satisfies the target accuracy, namely 95% of the original model; (3) **Horizontal Model Partition (HMP)**: we run the original vision transformer on all edge devices by splitting among layers for model parallelism; (4) **Vertical Model Partition (VMP)**: we run the original vision transformer on all edge devices by splitting within layers for tensor parallelism.

B. End-to-End Performance

1) Inference Latency: As shown in Figure 6a, our MDP outperforms all other baselines by $5.2\times \sim 2.1\times$. MDP can leverage all edge devices to run different sub-models in parallel with the limited communication to scatter the image data and reduce the final outputs. And all sub-models can be adapted to the heterogeneous capabilities of different edge devices with smaller image and model sizes. However, OVT and PVT can only run on the most powerful edge device. And PVT can only reduce about 50% costs of the original one. Both HMP and VMP make physical partitions on the inference workloads for the distributed edge inference. But the HMP executes sequentially on different devices, which cannot improve the latency of individual samples. And VMP suffers from the communication problem due to the global dependency and large feature size of vision transformers.

2) Inference throughput: Figure 6b shows that our MDP also have significant advantages over other baselines in the throughput. MDP can have $5.1\times \sim 1.7\times$ larger throughput than the others. Except for HMP, all other baselines keep the same trend in the latency experiment. HMP can better utilize all edge devices for better throughput by fulfilling the pipeline with enough inference data. Therefore, it can achieve the best throughout in all baselines, despite its relatively high latency.

3) Prediction Accuracy: Figure 7 illustrates that our MDP achieves comparable accuracy with the original vision transformer in both datasets, and it has considerable advantages over PVT. It is because virtual stacking can bring in extra information for better error correction. And supervised learning works as the auxiliary task for better feature learning and bagging ensemble. With different image sizes and model sizes, sub-models can build up feature pyramids and improve ensemble diversity. And the virtual stacking works as the regularization to improve the boosting ensemble. Therefore, all small sub-models can work together to maintain the accuracy of the large vision transformer. Compared with OVT, the PVT only generates a single smaller model, so it has a larger accuracy drop. We omit HMP and VMP for accuracy because the physical partition has the same accuracy as OVT.

Table I: The latency breaking-down for all methods

Method	comp. time (ms)	comm. time (ms)	comm. size (MB)
OVT	180.5	0	0
PVT	124.5	0	0
HMP	281.1	25	3.1
VMP	76.1	226	28.3
Ours	58.5	0.8	0.059

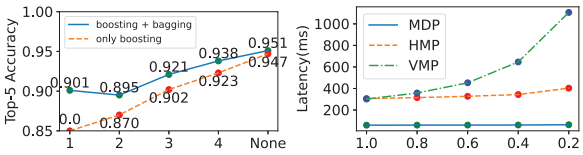
C. Deep-diving Experiments

1) Latency Breaking-down: Table I shows the latency breakdown for all the methods. Our MDP has the smallest communication cost in the distributed inference methods, thanks to the small size of image data and final outputs. VMP has achieved good speed up in the computation, but its expensive all-gather communications in every layer slow down the inference. HMP also has considerable device communications at every layer splitting point, and it even improves the computation time for the individual data because of the sequential computation on all different devices. Both OVT and PVT have no communication because of running on only one device. Compared with MDP 's sub-models in the ensemble, the single small model in PVT has limited rooms to be compressed without more accuracy loss.

2) Ablation Study in Accuracy: To verify the effectiveness of our designs to maintain accuracy, we conduct the ablation study shown in Table II. Without layer pruning for shallow sub-models running on extremely weak devices, MDP can improve some accuracy because of the deeper and stronger sub-models. When we keep the same small image size in all sub-models, it has some accuracy loss because of no feature pyramid of different scales and fewer ensemble diversities. Removing the virtual stacking also cause accuracy loss, because it brings in extra information about the previous sub-models for better error correction. And if we directly train the sub-models separately for the bagging ensemble instead of boosting ensemble, they have less accuracy improvement over

Table II: Ablation Study in Accuracy

	ImageNet(top-5)	Indoor(top-1)
original MDP	0.951	0.821
w/o layer pruning	0.957	0.824
w/o different image size	0.943	0.817
w/o virtual stacking	0.936	0.811
replace boosting with naive bagging	0.912	0.805
the best sub-model alone	0.889	0.796



(a) Time-out (b) Network Bandwidth(Gbs)

Figure 8: (Left): bagging ensemble to handle time-out exception happens on the N-th sub-model. (Right):Influences of different network bandwidth

the single model. The reason is all sub-models have small diversity due to being pruned from the same model in the same way. And there is a large gap between MDP and the best stand-alone sub-model because the single sub-model has limited model capability compared with the OVT and MDP .

3) Extra Bagging Ensemble for Time-out Exception: As shown in Figure 8a, we make experiments to verify the effectiveness of the extra added bagging ensemble. It illustrates that the added bagging term always helps get better accuracy, no matter which sub-model has the time-out exception. Especially if the first sub-model has a time-out problem, the boosting ensemble alone cannot even predict. But MDP employs the bagging ensemble of the following sub-models to avoid the waste of computations and improves the accuracy a little. Due to different other training losses, different training subsets and image sizes, and different layer numbers, our sub-models have higher diversities than the direct bagging ensemble.

4) Influences of Network Bandwidth: Because LAN can be shared by many other tasks, we show how different network bandwidths affect the latency of all distributed inference methods in Figure 8b. The network bandwidth variation has the smallest inferences on our MDP , because of the limited transferring volume. VMP can suffer more from all-gather communication in every layer. Compared with VMP, HMP is less affected when only communication in the layer splitting.

VII. Related Work

With the increasing size of deep models, there have been various works having explored distributed machine learning for acceleration. It employs multiple devices working in parallel to speed up either the training phase or inference phase. However, the cooperation of different devices brings in extra communication costs. The related previous works have studied in how to realize efficient distributed machine learning from different perspectives in the both following phases:

Training Phase: it usually distributes different data samples and model parts over different devices to run in parallel. The data parallelism requires the gradient allreduce among all devices, resulting in expensive global communication during the synchronization. Generally speaking, there are a wide range of approaches to improve the efficiency for distributed training, including but not limited to: 1) reducing the traffic volume in every iteration by gradient compression [37], [38] and frozen layers [39]; 2) relaxing the global synchronization requirement [40], [41], [42]; 3) overlapping communication over computation [43]; 4) employing fast RDMA network [44], [45], [46]; 5) leveraging new programmable devices like smart switches and FPGAs [47], [48], [49]; 6) optimizing network transferring by using congestion control [50], [51], [52] and flow scheduling [53], [54], [55], [56].

Inference Phase Some previous works also explore how to parallelize the model inference [57], [58] for the efficient deployment. Initially, previous works study how to compress the large model into a single small model, including quantization [9], [10], knowledge distillation [11], [12], model pruning [13], [14], and patch slimming [15]. However, they ignore the possibility to leverage multiple different devices for distributed model inference. Then some works attempt to offload the inference to the powerful remote servers with GPUs. Model cascade [1], [59], [60], [61] adopts the idea of conditional computation. They usually first run a small but weak model in the near-end edge devices. If the model confidence is low, they run the lager model or image on the remote powerful servers. However, it leads to not only high transferring latency over Internet [2], [5], [6] but also raising privacy concerns [62], [5], [6]. Recently, some other works explore the distributed inference on multiple local edge devices in the similar way of model parallelism. Existing distributed edge inference focuses on making proper physical vertical or horizontal partitions on the model architecture [26], [27], [2].

VIII. Conclusion

We propose the novel MDP to decompose the implicitly ensembled vision transformer into the explicitly boosting ensemble of different sub-models for efficient distributed edge inference. Virtual stacking can physically decouple all sub-models from each other while reserving information within themselves to maintain accuracy. With different image sizes and model sizes, all sub-models can have suitable workloads for heterogeneous devices and improve the diversity of the ensemble. We further leverage the bagging ensemble to handle unexpected time-out sub-models. We have conducted comprehensive experiments to verify its advantages in terms of latency, throughput and accuracy.

Acknowledge

This work is supported in part by National Key R&D Program of China (2022YFB4500302). We thank Dr. Huangxun Chen for the helpful discussion and the anonymous reviewers for their constructive feedback and suggestions.

References

[1] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *INFOCOM*, 2019.
[2] C. Hu and B. Li, "Distributed inference with deep learning models across heterogeneous edge devices," in *INFOCOM*. IEEE, 2022.
[3] Y. Li, T. Zeng, X. Zhang, J. Duan, and C. Wu, "Tapfinger: Task placement and fine-grained resource allocation for edge machine learning," in *INFOCOM*, 2023.
[4] C. Quan, W. Kaijia, G. Song, C. Zhipeng, and Z. Albert, "Latency-optimal pyramid-based joint communication and computation scheduling for distributed edge computing," in *INFOCOM*, 2023.
[5] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, 2019.
[6] W. Ren, Y. Qu, C. Dong, Y. Jing, H. Sun, Q. Wu, and S. Guo, "A survey on collaborative dnn inference for edge intelligence," *arXiv*, 2022.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021.

[8] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *CVPR*, 2022.

[9] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, "Post-training quantization for vision transformer," *NeurIPS*, 2021.

[10] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun, "Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization," in *ECCV*, 2022.

[11] Z. Hao, J. Guo, D. Jia, K. Han, Y. Tang, C. Zhang, H. Hu, and Y. Wang, "Learning efficient vision transformers via fine-grained manifold distillation," *NeurIPS*, 2022.

[12] S. Yu, T. Chen, J. Shen, H. Yuan, J. Tan, S. Yang, J. Liu, and Z. Wang, "Unified visual transformer compression," *arXiv*, 2022.

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

[14] Z. Kong, P. Dong, X. Ma, X. Meng, M. Sun, W. Niu, B. Ren, M. Qin, H. Tang, and Y. Wang, "Hfsp: A hardware-friendly soft pruning framework for vision transformers."

[15] Y. Tang, K. Han, Y. Wang, C. Xu, J. Guo, C. Xu, and D. Tao, "Patch slimming for efficient vision transformers," in *CVPR*, 2022.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv*, 2018.

[17] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[19] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," *NeurIPS*, 2016.

[20] F. Huang, J. Ash, J. Langford, and R. Schapire, "Learning deep resnet blocks sequentially using boosting theory," in *ICML*, 2018.

[21] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *CVPR*, 2017.

[22] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontañón, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," in *NeurIPS*, 2020.

[23] A. Fan, E. Grave, and A. Joulin, "Reducing transformer depth on demand with structured dropout," *arXiv*, 2019.

[24] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," *arXiv*, 2018.

[25] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *NeurIPS*, vol. 32, 2019.

[26] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *TCAD*, 2018.

[27] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, 2020.

[28] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv*, 2020.

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

[30] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *ICLR*, 2018.

[31] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv*, 2019.

[32] A. D. I. Pytorch, "Pytorch," 2018.

[33] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv*, 2019.

[34] Facebook, "Gloo," https://github.com/facebookincubator/gloo.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[36] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *CVPR 2009*.

[37] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *ICLR 2018,*.

[38] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: communication-efficient SGD via gradient quantization and encoding," in *NeurIPS*, 2017.

[39] Y. Wang, D. Sun, K. Chen, F. Lai, and M. Chowdhury, "Egeria: Efficient DNN training with knowledge-guided layer freezing," in *Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys 2023, Rome, Italy, May 8-12, 2023*, 2023.

[40] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing, "More effective distributed ML via a stale synchronous parallel parameter server," in *NeurIPS 2013*.

[41] W. Wang, C. Zhang, L. Yang, K. Chen, and K. Tan, "Addressing network bottlenecks with divide-and-shuffle synchronization for distributed DNN training," in *IEEE INFOCOM 2022*.

[42] X. Wan, K. Xu, X. Liao, Y. Jin, K. Chen, and X. Jin, "Scalable and efficient full-graph GNN training for large graphs," *Proc. ACM Manag. Data*, 2023.

[43] Y. Ma, H. Wang, Y. Zhang, and K. Chen, "Autobyte: Automatic configuration for optimal communication scheduling in DNN training," in *IEEE INFOCOM*, 2022.

[44] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng, T. Wang, W. Ling, K. Huo, P. An, K. Ji, S. Zhang, B. Xu, R. Feng, T. Ding, K. Chen, and C. Guo, "SRNIC: A scalable architecture for RDMA nics," in *NSDI 2023*.

[45] B. Yi, J. Xia, L. Chen, and K. Chen, "Towards zero copy dataflows using RDMA," in *Posters and Demos, SIGCOMM 2017*.

[46] Z. Ren, M. Fan, Z. Wang, J. Zhang, C. Zeng, Z. Huang, C. Hong, and K. Chen, "Accelerating secure collaborative machine learning with protocol-aware rdma," in *Security 2024*.

[47] C. Zeng, L. Luo, T. Zhang, Z. Wang, L. Li, W. Han, N. Chen, L. Wan, L. Liu, Z. Ding, X. Geng, T. Feng, F. Ning, K. Chen, and C. Guo, "Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing," in *NSDI 2022*.

[48] J. Zhang, X. Cheng, W. Wang, L. Yang, J. Hu, and K. Chen, "FLASH: towards a high-performance hardware acceleration architecture for cross-silo federated learning," in *NSDI 2023*.

[49] C. Zeng, L. Luo, Q. Ning, Y. Han, Y. Jiang, D. Tang, Z. Wang, K. Chen, and C. Guo, "FAERY: an fpga-accelerated embedding-based retrieval system," in *OSDI 2022*.

[50] Y. Ma, H. Tian, X. Liao, J. Zhang, W. Wang, K. Chen, and X. Jin, "Multi-objective congestion control," in *EuroSys '22*.

[51] J. Zhang, W. Bai, and K. Chen, "Enabling ECN for datacenter networks with RTT variations," in *CoNEXT 2019,*.

[52] J. Zhang, C. Zeng, H. Zhang, S. Hu, and K. Chen, "Liteflow: towards high-performance adaptive neural networks for kernel datapath," in *SIGCOMM 2022*.

[53] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, and C. Tian, "Information-agnostic flow scheduling for commodity data centers," in *NSDI 2015*.

[54] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: toward automatically identifying and scheduling coflows in the dark," in *SIGCOMM 2016*.

[55] W. Li, C. Zeng, J. Hu, and K. Chen, "Towards fine-grained and practical flow control for datacenter networks," in *ICNP 2023*.

[56] J. Hu, C. Zeng, Z. Wang, J. Zhang, K. Guo, H. Xu, J. Huang, and K. Chen, "Enabling load balancing for lossless datacenters," in *ICNP 2023*.

[57] W. Wang, Y. Zhang, S. Yan, Y. Zhang, and H. Jia, "Parallelization and performance optimization on face detection algorithm with opencl: A case study," *Tsinghua Science and Technology 2012*.

[58] W. Wang, Y. Zhang, G. Long, S. Yan, and H. Jia, "CLSIFT: an optimization study of the scale invariance feature transform on gpus," in *HPCC/EUC 2013,*.

[59] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *INFOCOM*, 2020.

[60] Y. Wang, W. Wang, D. Liu, X. Jin, J. Jiang, and K. Chen, "Enabling edge-cloud video analytics for robotics applications," *IEEE Trans. Cloud Comput.*, 2023.

[61] Y. Wang, K. Chen, H. Tan, and K. Guo, "Tabi: An efficient multi-level inference system for large language models," in *EuroSys 2023*, 2023.

[62] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *ICDCS 2017*.