PipeEdge: Pipeline Parallelism for Large-Scale Model Inference on Heterogeneous Edge Devices

Yang Hu*, Connor Imes†, Xuanang Zhao‡, Souvik Kundu*, Peter A. Beerel*, Stephen P. Crago†, John Paul Walters†
*Ming Hsieh Department of Viterbi School of Engineering, University of Southern California
Email: {yhu21003, souvikku, pabeerel}@usc.edu
†Information Sciences Institute, University of Southern California
Email: {cimes, crago, jwalters}@isi.edu
‡Google
Email: xuanangz@google.com

***Abstract*—Deep neural networks with large model sizes achieve state-of-the-art results for tasks in computer vision and natural language processing. However, such models are too compute- or memory-intensive for resource-constrained edge devices. Prior works on parallel and distributed execution primarily focus on training—rather than inference—using homogeneous accelerators in data centers. We propose PipeEdge, a distributed framework for edge systems that uses pipeline parallelism to both speed up inference and enable running larger, more accurate models that otherwise cannot fit on single edge devices. PipeEdge uses an optimal partition strategy that considers heterogeneity in compute, memory, and network bandwidth. Our empirical evaluation demonstrates that PipeEdge achieves $11.88\times$ and $12.78\times$ speedup using 16 edge devices for the ViT-Huge and BERT-Large models, respectively, with no accuracy loss. Similarly, PipeEdge improves throughput for ViT-Huge (which cannot fit in a single device) by $3.93\times$ over a 4-device baseline using 16 edge devices. Finally, we show up to $4.16\times$ throughput improvement over the state-of-the-art PipeDream when using a heterogeneous set of devices.**

***Index Terms*—deep learning, parallel execution, edge devices, large model inference**

I. Introduction

In recent years deep neural network (DNN) model sizes have increased significantly to provide improved accuracy [1]. For example, large transformer-based models achieve state-of-the-art accuracy in various computer vision (CV) [2], [3] and natural language processing (NLP) tasks [4], [5], but pose significant challenges for resource-constrained deployment, especially at the edge where resource-constrained devices exist in close proximity to data sources [6]–[9]. In particular, the vision transformer model (ViT-Large) [2] has $\sim$307M parameters and requires about 478B FLOPs to perform inference on one image [10]. More importantly, the super-linear growth of the models of $\sim 240\times$/2-years [11], coupled with the slowdown of Moore's law has created a significant memory bottleneck for their resource-efficient deployment. Figure 1 depicts the limitations of ViT models' deployment in terms of limited throughput and out-of-memory on an RCC-VE [12] and MinnowBoard edge device [13], respectively.

Various methods have been proposed to address large model inference challenges on edge devices, including model compression [14]–[19], adaptive inference [20], [21], and neural architecture search [22]. These approaches reduce the number of required computation operations, but at the cost of reduced accuracy, and most are limited to running on a single device. In contrast, collaborative edge computing leverages underutilized or idle distributed edge resources, *enabling multiple devices to collaborate and exceed the capability of any single device for large-scale model inference* [23], [24]. Pipeline parallelism has proven to be an effective technique for accelerating large data center models [25]. This method exploits model parallelism by partitioning models into multiple stages, which can *accelerate processing without accuracy loss* by utilizing additional distributed resources. Research on pipeline parallelism has focused on data center scenarios with high interconnect bandwidth and homogeneous accelerators like graphics processing units (GPUs) and tensor processing units (TPUs) [26]–[28]. Several frameworks consider pipeline parallelism for limited heterogeneity in data centers, e.g., heterogeneous communication topologies with homogeneous GPUs [29], [30] or heterogeneous GPU clusters with homogeneous networks [31]. Torchpipe [32] provides an automatic balancing strategy for large models, but only for the single-node scenario and does not claim optimality. Finding an optimal partition strategy that accounts for heterogeneity in compute, memory, *and* communication is critical for edge deployment scenarios, yet remains a largely open problem.

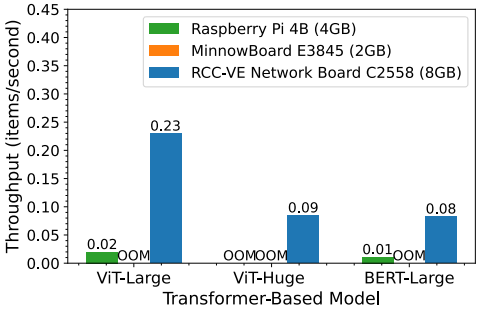

Fig. 1. Transformer-based model performance on three edge devices. Throughput is measured in images/second for ViT and sequences/second for BERT. `OOM` indicates the model does not fit in memory.

We address these challenges with PipeEdge, a distributed inference framework that exploits pipeline parallelism to improve inference performance on heterogeneous edge devices and networks. This paper makes the following contributions:

- A distributed pipeline parallelism framework to accelerate large-scale model inference for heterogeneous edge computing without accuracy loss.

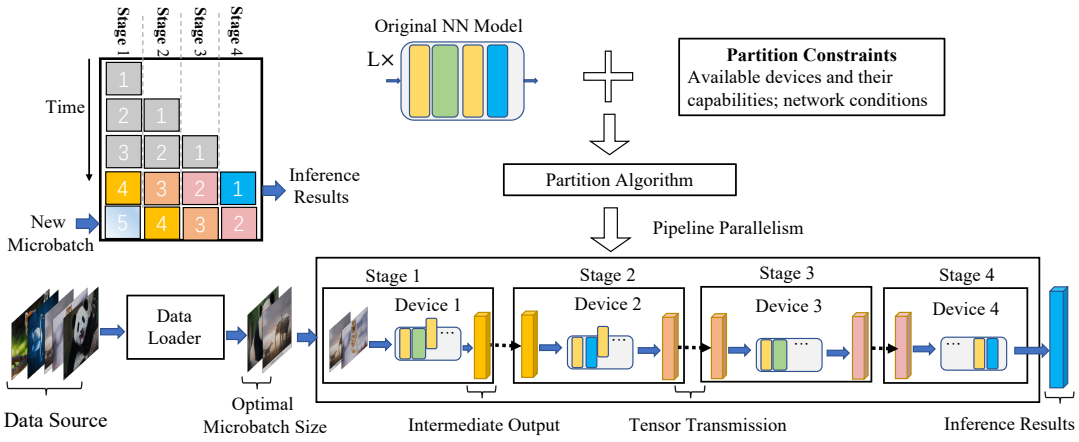

Fig. 2. PipeEdge system design overview. PipeEdge automatically partitions models between edge devices, subject to device, network, and model constraints. Intermediate results then are transmitted between pipeline stages. (Figure best viewed in color.)

- A dynamic programming (DP) algorithm to determine the optimal partition scheduling strategy for heterogeneous devices and communication channels.
- A detailed experimental evaluation on a real edge testbed, demonstrating throughput improvements up to $12.78\times$ in a 16-device homogeneous cluster and $4.16\times$ over the state-of-the-art PipeDream on a heterogeneous cluster[1].

II. Background and Motivation

High-accuracy machine learning models are traditionally designed for use on server-class systems in cloud environments and data centers. However, due to various reasons including privacy [33], inference operations are moving closer to the edge, where devices are closer to data sources, which can also improve response times. Edge systems differ from their data center counterparts due to size, weight, and power constraints – they typically have considerably less memory (e.g., tens of MB to several GB), slower compute (e.g., fewer and slower CPUs, and less powerful—if any—accelerators like GPUs), and rely on slower wireless communication (e.g., tens of Kbps to hundreds of Mbps, using WiFi or 4G/5G radio) rather than high-speed wired interconnects. Edge systems are also relatively diverse and dispersed, and thus heterogeneous in their compute, memory, and communication capabilities.

Achieving acceptable inference performance is challenging for resource-limited edge devices. Classical model compression techniques, including pruning [34], [35], quantization [16], low-rank approximation [36], and knowledge distillation [14] can shrink neural network model sizes to potentially accelerate large models, but often require iterative retraining and a full-precision pre-trained model to avoid significant *accuracy loss*. Additionally, these methods generally consider only on a single compute node. Distributed edge computing scenarios such as vehicular edge computing (VEC) for internet of vehicles [37], wireless-connected AI-enabled sensors [38], and smart home systems [39], [40], in contrast, often include a large number of resource-limited devices that can be utilized collaboratively.

[1]Source code is available at: https://github.com/usc-isi/PipeEdge

Pipeline parallelism partitions a neural network model into multiple stages, where each stage consists of a consecutive set of layers in the original model [26], [29]. Each stage is then assigned to a worker, thus parallelizing the model training or inference pipeline. Each worker sends its output data only to the next worker, which avoids collective communication and synchronization between all workers. The pipeline input minibatch is split into multiple chunks of equal size called microbatches [25]. The microbatch size affects pipeline performance, with the optimal size depending on multiple factors including the model characteristics and the number of pipeline stages [10]. Pipelining can also overlap computation and communication to improve performance [29].

While pipeline parallelism has proven to be effective for distributed training on accelerators in data centers [25], [26], [28], the aforementioned edge computing characteristics present challenges to using distributed pipeline parallelism for large-model inference at the edge. Model partition methods developed for homogeneous data center clusters tend to perform poorly in heterogeneous edge environments. A new pipeline parallelism framework is needed to overcome these challenges. In the next section, we introduce a framework for distributed edge clusters that uses heterogeneity-aware pipeline parallelism to improve inference performance and enable running larger—and more accurate—models than may otherwise be possible.

III. Parallelism for Edge Devices

A. PipeEdge System Design

Figure 2 presents the PipeEdge system design. First, partition constraints are provided to the partition algorithm which schedules an optimal layer-to-device partitioning (Section III-B). Partition constraints include the model properties and the available edge devices with their computation, memory and bandwidth capabilities. The PipeEdge runtime then deploys the pipeline stages to selected devices, where each device is only responsible for the inference of one part of the original model, as specified by the schedule. Pipeline input data is split into chunks called microbatches such that they fit into each edge device's memory and the size choice optimizes throughput (evaluated in Section V-D). After processing each microbatch,

TABLE I
SYMBOL DEFINITIONS

Symbol	Description
$\mathbb{T}, L, P_j, M_j$	The model with L layers. Each layer j has P_j parameters for transmission and requires M_j memory for execution.
$\mathbb{D}, D, m_v$	The list of D available devices. Each device v has memory capacity m_v.
$\mathbb{S}, S$	The list of S selected devices. Every device should participate in the inference.
$\mathbb{B}, b_{u,v}$	The list of bandwidths, where each device pair u, v have bandwidth $b_{u,v}$.
$\mathbb{R}$	The optimal mapping strategy.
$T_{\text{comp}}(l, u)$	The computation time for layers l on device u.
$T_{\text{comm}}(u, v, P_j)$	The communication time for transferring P_j parameters from device u to v.
$T_{\text{period}}(l, u, v, P_j)$	The maximum latency for executing layers l on device u and transferring P_j parameters to device v.
T_{opt}	The optimal pipeline stage time for maximum throughput.

the edge device transmits intermediate outputs to the device in the next pipeline stage. The device in the final stage produces the final result, which might then be transmitted to another host or used locally.

B. Partition Scheduling

We define a model $\mathbb{T}$ with L layers, inter-layer transmission data size P_j for each layer $j \in L$, and a list of heterogeneous devices $\mathbb{D}$ ($|\mathbb{D}| = D$) with different memory, computation, and communication capabilities. In heterogeneous communication, the bandwidth between a pair of devices u and v may be different than the bandwidth between a different pair of devices u' and v': $b_{u,v} \neq b_{u',v'}$. The optimal strategy $\mathbb{R}$ partitions the model $\mathbb{T}$ into S parts and allocates them to the selected devices $\mathbb{S} \subseteq \mathbb{D}$ ($|\mathbb{S}| = S \leq D$) to achieve maximal throughput and conform to device memory constraints.

We denote $T_{comp}(l, u)$ as the execution time for layers $l \subseteq \mathbb{T}$ on device u. $T_{comm}(u, v, P_j)$ is the time to communicate data P_j from u to v—the next device in the pipeline. Where $b_{u,v}$ is the bandwidth between u and v, T_{comm} is computed as:

$$T_{\text{comm}}(u, v, P_j) = \frac{P_j}{b_{u,v}} \tag{1}$$

An efficient pipeline implementation supports asynchronous communication, where computation and communication are overlapped. Thus, the maximum latency for the single device u can be calculated as:

$$T_{\text{period}}(l, u, v, P_j) = \max \begin{cases} T_{\text{comp}}(l, u) \\ T_{\text{comm}}(u, v, P_j) \end{cases} \tag{2}$$

Achieving the maximal throughput is equivalent to minimizing the execution time of the slowest stage, i.e., the largest $T_{\text{period}}(l, u, v, P_j)$, which we denote as T_{opt}. The pipeline partitioning problem can itself be partitioned. The optimal solution for partitioning the whole pipeline can be constructed from the optimal partitioning result for the sub-problem, which can be solved by dynamic programming (DP) methods.

To tackle this partition problem for heterogeneous clusters, we design a three-dimensional DP algorithm which records the state of processed layers, used devices, and the device in the last pipeline stage. Let $h(i, \mathbb{S}, u)$ denote the minimum time to process the first i layers using the set of devices $\mathbb{S} \subseteq \mathbb{D}$, where u is the next device to be used. $h(i, \mathbb{S}, u)$ is the optimal solution of the subproblem for i layers and $\mathbb{S}$ devices. The final optimal solution of this partition problem is the minimum $T_{\text{opt}} = h(L, \mathbb{S}, \varnothing)$.

The calculation of $h(j, \mathbb{S} \cup \{u\}, v)$ needs to use the optimal subproblem property, which is determined by the previous state $h(i, \mathbb{S}, u), 0 \leq i < j \leq L$, or the calculation time $T_{\text{period}}(\{i \to j\}, u, v, P_j)$ from i-th layer to j-th layer on the current device u. We further analyze these two situations:

- $h(j, \mathbb{S} \cup \{u\}, v) \leftarrow h(i, \mathbb{S}, u)$, the slowest pipeline stage for j units, is determined by the previous stage $h(i, \mathbb{S}, u)$. Since device u implements the current stage from the i-th to the j-th layer in the current pipeline, the used devices set for the next state $h(j, \mathbb{S} \cup \{u\}, v)$ should include the device u, i.e., $\mathbb{S} \cup \{u\}$. Parameters i and $u \in \mathbb{D} \setminus \mathbb{S}$ will be enumerated to find the optimal solution of the subproblem.
- $h(j, \mathbb{S} \cup \{u\}, v) \leftarrow T_{\text{period}}(\{i \to j\}, u, v, P_j)$, where device u is the slowest stage of the current pipeline candidate and limits the performance of the system. Similarly, device u and first i layers are enumerated to obtain the minimum value.

Thus, the state transition equation can be formulated as:

$$\begin{aligned} h(j, \mathbb{S}', v) &= \min_{\substack{0 \leq i < j \leq L \\ u, v \in \mathbb{D} \setminus \mathbb{S}}} \max \begin{cases} h(i, \mathbb{S}, u) \\ T_{\text{period}}(\{i \to j\}, u, v, P_j) \end{cases} \\ &= \min_{\substack{0 \leq i < j \leq L \\ u, v \in \mathbb{D} \setminus \mathbb{S}}} \max \begin{cases} h(i, \mathbb{S}, u) \\ T_{\text{comm}}(u, v, P_j) \\ T_{\text{comp}}(\{i \to j\}, u) \end{cases} \end{aligned} \tag{3}$$
$$\mathbb{S}' = \mathbb{S} \cup \{u\}$$

The first term inside the max is the minimum time for the first i layers using device set $\mathbb{S}$ and the next device u. The second term is communication time for transferring P_j data from u to v. The third term is the computation time for the last $j - i$ layers on device u. For initialization, $h(0, \emptyset, \varnothing)$ is set to 0.

Equation 3 calculates the optimal pipeline execution time. However, we need to obtain the selected devices and their order in the pipeline for the optimal strategy. Algorithm 1 describes the memoization technique pseudo-code to find the optimal time T_{opt} and the corresponding pipelining strategy.

The proposed algorithm's computational complexity is $O(2^D \times L^2 \times D^2)$, where D is the number of available devices and L is the number of layers. The 2^D factor is due to the assumption that all devices are distinct. For comparison, the naive brute force solution's search space is $\sum_{i=1}^{\min\{D,L\}} \frac{D!}{(D-i)!}\binom{L-1}{i-1} \gg D! \gg 2^D$, which has a much higher complexity. Moreover, in most scenarios, there should exist identical devices with the same computation and communication capabilities. The number of devices D can then be divided into N categories, where each category i has n_i devices ($\sum_{i=1}^{N} n_i = D$). This reduces the DP search space such that the computation complexity is then $O(\prod_{i=1}^{N}(n_i + 1) \times L^2 \times N^2)$. Consider the case where there are three device types ($N = 3$)

Algorithm 1 DP-based Pipeline Partition Strategy

Require:
$\mathbb{T}$: model with L layers, parameter set P, and memory requirements M;
$\mathbb{D}$: available devices with memory m_v for $v \in \mathbb{D}$;
$\mathbb{B}$: bandwidth between devices;
Ensure:
T_{opt}: optimal time for maximum throughput;
$\mathbb{R}$: specific strategy for the optimal time;

procedure PARTITION($\mathbb{T}, \mathbb{D}, \mathbb{B}$)
  Initial $h(i, \mathbb{S}, u) \leftarrow +\infty$ for all $i \in L, \mathbb{S} \subseteq \mathbb{D}, u \in \mathbb{D}$;
  Initial $h(0, \emptyset, \varnothing) \leftarrow 0$;
  Initial $answer \leftarrow \infty$;
  for $i = 0$ to $L - 1$ **do**
    for each subset $\mathbb{S} \subseteq \mathbb{D}$ **do**
      for each $u \in \mathbb{D} \setminus \mathbb{S}$ **do**
        for $j = i + 1$ to L **do**
          if $\sum_{k=i}^{j} M_k > m_u$ **then**
            Break;
          end if
          Calculate Eq.(3);
          Assign the Eq.(3) value to C;
          if $j == L$ **then**
            if $C < answer$ **then**
              $answer = C$;
              $index = (L, \mathbb{S}, u)$;
            end if
          else
            for each $v \in \mathbb{D} \setminus \mathbb{S} \setminus \{u\}$ **do**
              if $C < h(j, \mathbb{S} \cup \{u\}, v)$ **then**
                $h(j, \mathbb{S} \cup \{u\}, v) = C$;
                // Record the precursor
                $p(j, \mathbb{S} \cup \{u\}, v) = (i, u)$;
              end if
            end for
          end if
        end for
      end for
    end for
  end for
  // Find the optimal results
  Initial $T_{\text{opt}} \leftarrow +\infty$;
  while enumerate each subset $\mathbb{S} \subseteq \mathbb{D}$ **do**
    $T_{\text{opt}} = \min(h(L, \mathbb{S}, \varnothing), T_{\text{opt}})$;
  end while
  // Find the optimal strategy
  $(i, \mathbb{S}, u) = index$;
  Add $(i + 1 \rightarrow L, u)$ to $\mathbb{R}$;
  while $i > 0$ **do**
    $(i, u) = p(index)$;
    Add $(i + 1 \rightarrow index[0], u)$ to $\mathbb{R}$;
    $index = (i, \mathbb{S} \setminus u, u)$;
  end while
  return $T_{\text{opt}}, \mathbb{R}$
end procedure

TABLE II
PARTITIONING METHOD PERFORMANCE.

Algorithm	Time	Speedup
Brute force search	71 min	–
Naive dynamic programming	18.6 sec	229×
Category dynamic programming	0.01 sec	426,000×

and each type has the same number of devices, i.e., $n_1 = n_2 = n_3 = n$. Then the actual computational complexity is $O((n+1)^3 \times L^2 \times N^2) = O(9 \times (n+1)^3 \times L^2)$. Given $N = 3$ device types, where each type has $n = 3$ devices, we measure the execution time for these three methods for 48 partitionable layers on a 1.6 GHz Intel Core i5 CPU and present the results in Table II. The categorical approach's significantly better performance supports larger scale problems and can allow the partition algorithm to run more frequently.

C. Fine-grained Partition for Transformer Blocks

While PipeEdge's design is general enough to support different variants of DNNs including CNNs and MLPs, our evaluation focuses on transformer models, which we summarize here. Transformers were proposed to improve the effectiveness in learning dependencies between distant positions for sequence modeling tasks [4]. They have been widely used for NLP tasks [41] and recently extended to replace CNN models for performing complex CV tasks [2], [17]. In particular, the ViT models enjoy superior representation ability [42] and suffer less from positional invariance issues which are prevalent in conventional CNNs [43].

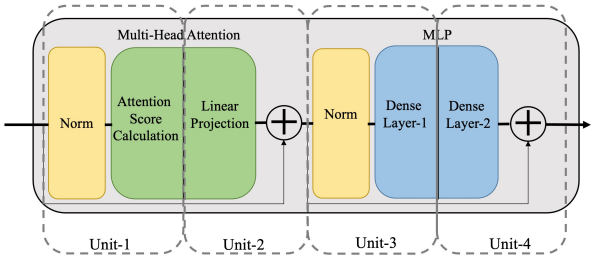

Fig. 3. Four partitioning units for one transformer block.

Transformer encoders include multiple transformer blocks with identical structures. GPipe [25] first proposed inter-layer pipeline parallelism and treats one layer as the smallest partition unit. However, finer-grained partitioning enables better workload balance for pipeline parallelism and can lead to greater scalability. We carefully analyze the computational complexity and structure of each layer and propose fine-grained partitioning for transformer-based models without changing the point-to-point communication pattern. The multi-head attention and MLP layers have the largest execution time compared to other layers. PipeEdge divides the multi-head attention layer into two parts: (1) calculation of the softmax attention scores and the multiplication of the attention score with the corresponding values; and (2) the linear projection of the self-attention output. Similarly, we divide the MLP layer into two dense layers. Layer normalization and residual connection operations are less computationally complex and require less execution time than the multi-head attention and MLP layers. Figure 3 visualizes partitioning transformer blocks into four components: (1) layer normalization with the first operation in the multi-head attention layer; (2) linear projection operation in the multi-head attention layer with the residual connection; (3) layer normalization with the first dense layer; and (4) the second dense layer with residual connections. The PipeEdge scheduling algorithm then simply treats these as distinct layers.

IV. EXPERIMENTAL SETUP

We conduct experiments on the Dispersed Computing Program Testbed (DCompTB) for edge computing platforms [44]. DCompTB exposes the two edge device types described in Table III. For evaluation, we first configure homogeneous edge clusters using both MinnowBoards and RCC-VEs, where each cluster is composed of identical devices and network bandwidth. For heterogeneous experiments, we mix device types and further increase heterogeneity by leveraging system software tools. We use the `cpulimit` tool to limit the CPU usage, the `ulimit` tool to limit the memory size on RCC-VEs, and the `tc` tool to vary network bandwidth between edge

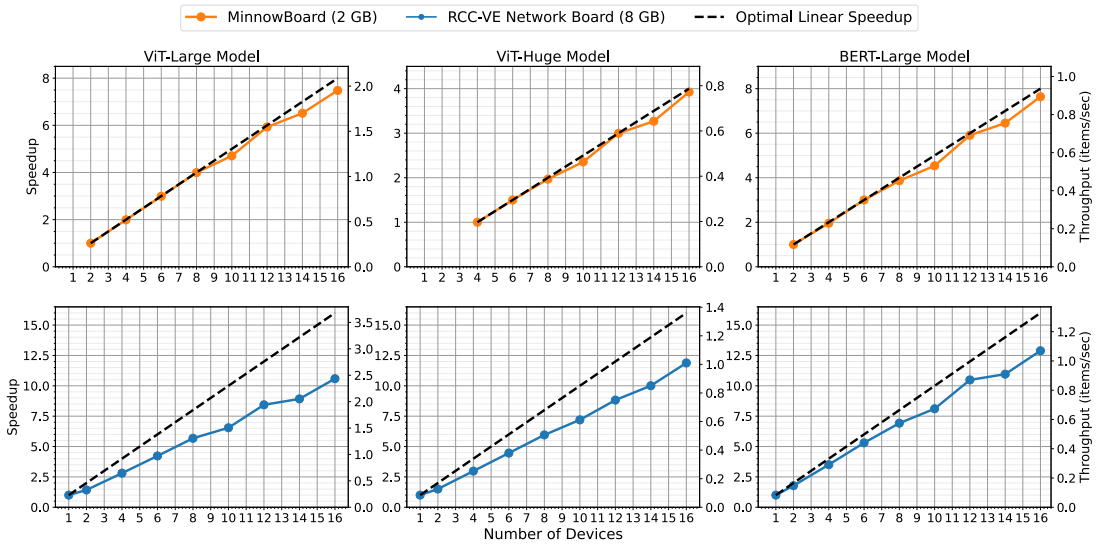

Fig. 4. PipeEdge's throughput performance for three transformer-based models on two homogeneous edge clusters. Models do not fit on a single MinnowBoard.

TABLE III
DCOMPTB DEVICE PROPERTIES.

Configuration	Value
Device name	MinnowBoard
Processors	4× Intel Atom E3845 @ 1.91 GHz
Memory size	2 GB
Max bandwidth	1 Gbps
Device name	RCC-VE Network Board
Processors	4× Intel Atom C2558 @ 2.4 GHz
Memory size	8 GB
Max bandwidth	1 Gbps

devices. In multiple edge computing scenarios, it is common to have tens of milliseconds latency [45], so we impose a fixed 20 ms latency when varying the bandwidth.

Each node runs Debian GNU/Linux 10 with Linux kernel 4.19.0-11-amd64. We use PyTorch 1.9.0 as the deep learning inference engine and the PyTorch RPC library with the TensorPipe backend as the distributed framework [46].

The definition, configuration, and implementation of the ViT and BERT models are from HuggingFace 4.10.0 [47][2]. We evaluate ViT models for the image classification task. Input images are from ImageNet 2012 [48] and resized after the embedding layer with a uniform input dimension to the models. We evaluate the BERT model for text classification on the SST-2 dataset from the General Language Understanding Evaluation (GLUE) [49] with a maximum sequence length of 512.

We adopt baselines from the state-of-the-art pipelines developed for large-scale models. We re-implement GPipe [25] with an even partitioning method for inference on the CPU as the baseline. Although PipeDream targets asynchronous parallel training, its partition method can be applied to inference by considering only the forward pass. PipeDream also provides open-source code for partitioning for inference [29]. PipeDream considers a hierarchical interconnect that represents a data center interconnect topology, but that does not model the ad hoc networks of edge systems. To apply PipeDream to the edge, we assume a one-level communication network and compare with its pipeline partitioning scheme. We choose the optimal microbatch size for GPipe and PipeDream and compare the system performance.

[2]While we limit our evaluations to transformer-based models, our approach is more general and can be easily extended to large convolution/MLP models, as the these models also reply on large numbers of MAC operations that can happen in parallel through partitioning of the GEMM-based operations.

V. EVALUATION

A. Runtime Performance Analysis

We first evaluate PipeEdge's performance on the 2 GB MinnowBoard and the 8 GB RCC-VE Network Board devices in homogeneous clusters. Figure 4 presents throughput on these clusters for up to 16 stages (devices).

The ViT-Large, ViT-Huge, and BERT-Large models are too big to fit in memory on a single MinnowBoard. We therefore use 2-stage, 4-stage, and 2-stage throughput as the speedup baselines for each model, respectively. With 16 MinnowBoards, PipeEdge achieves 1.95 images per second throughput, which is a 7.48× speedup over the 2-stage baseline for the ViT-Large model (optimal speedup is 16/2=8). For the ViT-Huge model, PipeEdge achieves 0.77 images per second throughput with 16 MinnowBoard devices, which is a 3.93× speedup over the 4-stage baseline (optimal speedup is 16/4=4). For the BERT-Large model, PipeEdge achieves 0.89 sequences per second, which is a 7.64× speedup relative to the 2-stage baseline (optimal speedup is 16/2=8).

We achieve similar scalability on the RCC-VE devices, where the aforementioned models fit in memory, allowing for a single-device baseline. With 16 RCC-VEs, PipeEdge achieves 2.43 and 1.01 images per second throughput for the ViT-Large and ViT-Huge models, which are 10.59× and 11.88× speedups, respectively. With the BERT-Large model, PipeEdge achieves 1.05 sequences per second throughput and 12.78× speedup with 16 devices.

PipeEdge achieves significant—in many cases, nearly linear—performance improvements for the three transformer-based

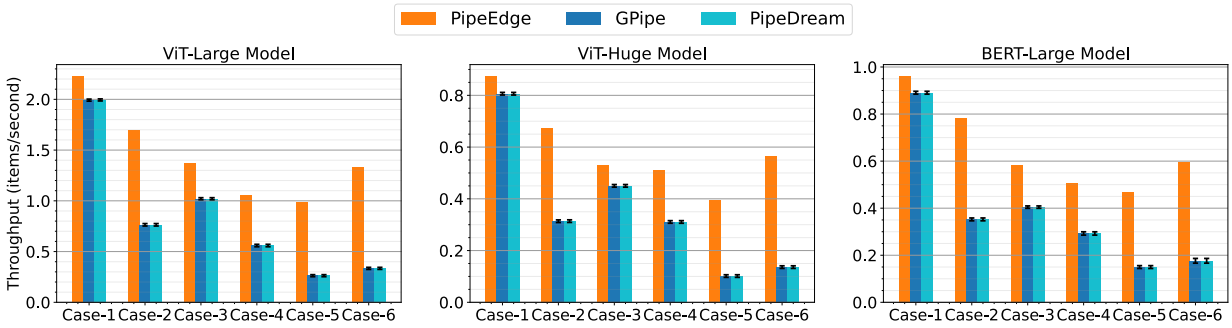

Fig. 5. Heterogeneous cluster throughput. For GPipe and PipeDream, average throughput and variance are shown for 10 random device orders.

TABLE IV
HETEROGENEOUS CLUSTER CONFIGURATIONS.

Case	Devices	CPU	Memory	Bandwidth
1	8×RCC-VE	100%	8 GB	1 Gbps
	8×MinnowBoard	100%	2 GB	1 Gbps
2	4×RCC-VE	100%	8 GB	1 Gbps
	4×RCC-VE	75%	4 GB	1 Gbps
	4×RCC-VE	25%	4 GB	1 Gbps
	4×MinnowBoard	100%	2 GB	1 Gbps
3	8×RCC-VE	100%	8 GB	40 Mbps
	8×MinnowBoard	100%	2 GB	10 Mbps
4	4×RCC-VE	100%	8 GB	30 Mbps
	4×RCC-VE	100%	8 GB	20 Mbps
	4×MinnowBoard	100%	2 GB	10 Mbps
	4×MinnowBoard	100%	2 GB	5 Mbps
5	3×RCC-VE	100%	8 GB	50 Mbps
	8×RCC-VE	10%	4 GB	20 Mbps
	5×MinnowBoard	100%	2 GB	30 Mbps
6	2×RCC-VE	100%	8 GB	100 Mbps
	3×RCC-VE	75%	4 GB	60 Mbps
	4×RCC-VE	50%	4 GB	40 Mbps
	3×RCC-VE	25%	4 GB	20 Mbps
	2×RCC-VE	10%	4 GB	10 Mbps
	2×MinnowBoard	100%	2 GB	80 Mbps

models on homogeneous clusters of both edge device types. These results demonstrate PipeEdge's effectiveness for large-scale models, especially those that otherwise cannot fit on single devices.

B. Heterogeneous Clusters

Compared to data centers, devices in edge environments are more heterogeneous in computation, memory, and communication capabilities. To further increase heterogeneity in our evaluation environment, we throttle RCC-VE CPUs to reduce inference performance and cap available memory. We also vary the maximum bandwidth for both device types to emulate different network link capacities. Table IV presents six device cluster configurations with increasing heterogeneity.

We compare PipeEdge with GPipe and PipeDream on these clusters. GPipe and PipeDream do not specify device mapping order, so we test them with 10 random device orders and measure average performance and variance. Since the computation is evenly distributed across transformer blocks [25], The GPipe and PipeDream partitioning methods degenerate to even partitioning for these clusters, due to lack of ability to handle device heterogeneity. Figure 5 presents the experimental results. PipeEdge outperforms GPipe and PipeDream in every heterogeneous case.

Cases 1 and 2 exhibit compute and memory heterogeneity. PipeEdge achieves 2.23 and 1.69 images per second with ViT-Large and 0.88 and 0.67 images per second with ViT-Huge, respectively. GPipe and PipeDream only achieve 1.99 and 0.76 images per second with ViT-Large and 0.81 and 0.31 images per second with ViT-Huge. PipeEdge also achieves the best BERT-Large performance – 0.96 and 0.78 sequences per second. GPipe and PipeDream show performance degradation with BERT-Large for different device orders – 0.89 and 0.35 sequences per second, respectively. PipeEdge clearly demonstrates better performance when compute and memory capabilities are heterogeneous.

Case 3 has the same compute and memory resources as Case 1, but with less communication bandwidth. Case 4 introduces further bandwidth heterogeneity with the same compute and memory resources. In both cases, PipeEdge achieves the best performance for ViT-Large and ViT-Huge, and with fewer devices than GPipe and PipeDream. In Case 3 for both ViT-Large and ViT-Huge, PipeEdge selects 8 devices with 40 Mbps bandwidth and one device with 10 Mbps bandwidth as the last stage, achieving 1.37 and 0.53 images per second throughput, respectively. In Case 3 for both ViT-Large and ViT-Huge, GPipe and PipeDream both use 16 devices, achieving only 1.02 and 0.44 images per second throughput. In Case 4, PipeEdge selects 7 devices for ViT-Large and 9 devices for ViT-Huge to achieve throughput of 1.05 and 0.51 images per second, respectively; GPipe and PipeDream only achieve 0.55 and 0.31 throughput for ViT-Large and ViT-Huge. In Case 4, PipeEdge shows 1.90× and 1.54× speedup compared to PipeDream for ViT-Large and ViT-Huge, respectively. In Cases 3 and 4 for BERT-Large, PipeEdge achieves the best throughput of 0.58 and 0.49 images per second. GPipe and PipeDream suffer significant performance degradation due to the limited bandwidth, which they do not account for – GPipe and PipeDream only achieve 0.35 and 0.29 sequences per second for BERT-Large. For Case 3 and 4, PipeEdge achieves 1.45× and 1.65× speedup compared with GPipe and PipeDream. These two cases demonstrate the effectiveness of PipeEdge's partitioning strategy for heterogeneous networks.

Cases 5 and 6 mix the heterogeneity of devices and networks. In Case 5, we add 8 extremely resource-constrained devices with CPUs at 10% capacity and 20 Mbps bandwidth. PipeEdge

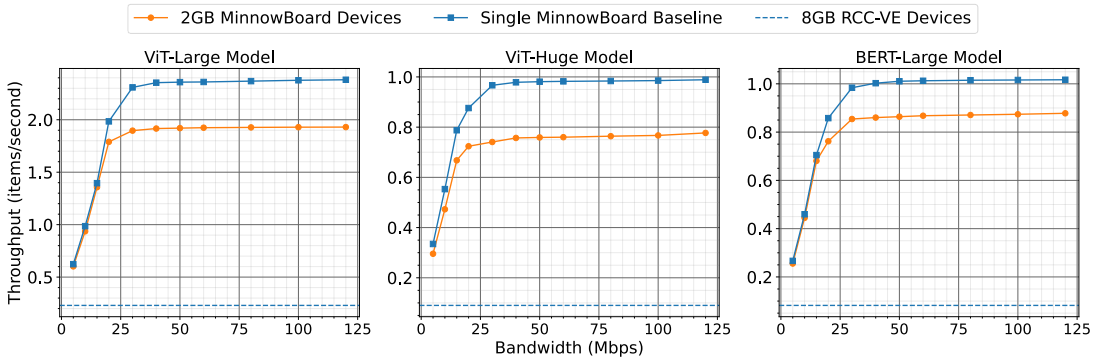

Fig. 6. The relationship between bandwidth and system throughput.

achieves the best throughput with 0.99, 0.39 images per second, and 0.47 sequences per second using 7 devices for ViT-Large, ViT-Huge, and BERT-Large. In Case 5 for ViT-Large and ViT-Huge, GPipe and PipeDream achieve 0.26 and 0.10 images per second; for BERT-Large model, they achieve 0.15 sequences per second. In Case 5, PipeEdge achieves 3.75×, 3.84×, and 3.13 speedup relative to PipeDream's average throughput for ViT-Large, ViT-Huge, and BERT-Large, respectively. Case 6 shows a scenario with 6 devices types, weighted toward devices with medium performance. In this case, PipeEdge uses 12, 14, and 12 devices to achieve 1.33, 0.57, and 0.59 items per second for these three transformer-based models; GPipe and PipeDream only achieve 0.33, 0.14, and 0.17 items per second, respectively. PipeEdge achieves speedup of 3.98×, 4.16× and 3.47× for ViT-Large, ViT-Huge, and BERT-Large compared to PipeDream's average throughput. Cases 5 and 6 demonstrate PipeEdge's ability to schedule around low-performance devices and map partitions to achieve the best throughput.

PipeEdge performs significantly better than the GPipe and PipeDream partition methods on all six heterogeneous clusters. Unlike GPipe and PipeDream, PipeEdge successfully avoids the lowest-performing devices by considering multiple factors when exploiting pipelining to improve performance.

C. Bandwidth Impact

To evaluate the relationship between system performance and bandwidth, we vary the bandwidth between all devices from 120 Mbps to 5 Mbps. We test with 16 pipeline stages using the ViT-Large, ViT-Huge, and BERT-Large models. Figure 6 shows that performance does not decrease significantly until bandwidth drops below 30 Mbps for these models, supporting PipeEdge's feasibility for practical edge applications. Reducing bandwidth from 30 Mbps to 5 Mbps shows a roughly linear performance decline, yet is still faster than the single-device baseline. At 5 Mbps with RCC-VE devices, PipeEdge still achieves 2.69×, 3.67×, and 3.14× speedups for ViT-Large, ViT-Huge, and BERT-Large compared to the single device baseline. These results demonstrate that PipeEdge is still effective for large-scale models under limited network conditions.

D. Microbatch Size Impact

As in other pipeline frameworks, performance is affected by the microbatch size. Figure 7 demonstrates the relationship between microbatch size and throughput. For a 2-stage pipeline, the maximum throughput of the even partitioning method in

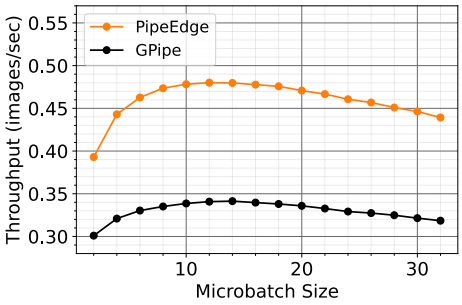

Fig. 7. Microbatch size vs. throughput for ViT-Base on MinnowBoards. The model is partitioned into 2 stages.

GPipe is around 0.34 images per second with a microbatch size 12. Increasing the microbatch size to about 14 results in a significant throughput improvement over smaller sizes, which is mainly due to the increasing CPU utilization as the microbatch size increases. Beyond this size, the throughput begins to decline because larger microbatch sizes reduce the efficiency of the pipeline parallelism. PipeEdge exhibits a similar pattern, achieving a maximum throughput of 0.48 images per second with a microbatch size of 12. The fine-grained partitioning method in PipeEdge achieves more efficient CPU utilization than GPipe's even partitioning. To give a fair comparison of throughput, we use the optimal microbatch sizes for PipeEdge, GPipe, and PipeDream for all other experiments in our evaluations.

Figure 8 compares PipeEdge's performance with a microbatch size of 1 and its optimal microbatch size with ViT-Large and varying RCC-VE device counts. PipeEdge still shows a 9.34× speedup on 16 devices with a small microbatch size, compared to the 10.59× obtained with the larger microbatch size. Smaller microbatches can reduce the output latency (and hence the response time) for applications, but at a reduced throughput. This experiment demonstrates that PipeEdge is effective for different microbatch sizes, which is important for latency-sensitive applications.

E. PipeEdge with Model Compression

Model compression shrinks model sizes to reduce compute cost and potentially accelerate training [50] and inference [14], [15], but often at the cost of reduced accuracy. These approaches are important complementary strategies to PipeEdge

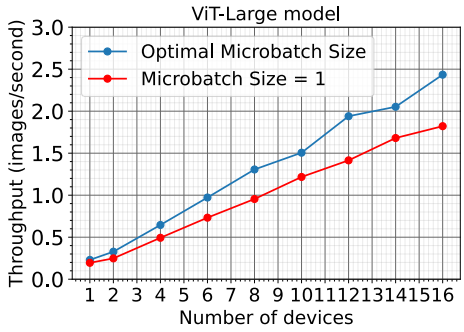

Fig. 8. The throughput of optimal microbatch size vs. microbatch size is 1 for ViT-Large on RCC-VE boards.

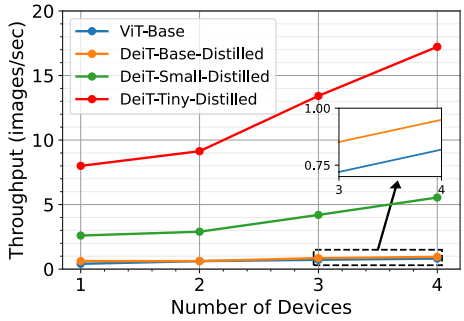

Fig. 9. DeiT distilled model throughput on RCC-VE devices.

for improving performance on resource-constrained platforms. For example, compared to the base ViT model, DeiT-Tiny and DeiT-Small use distillation to achieve similar ImageNet top-1 accuracy with compressed models of up to $17.2\times$ and $3.9\times$, respectively [51]. To demonstrate the efficacy of using compressed models with PipeEdge, we evaluate DeiT-Base, Small, and Tiny on up to 4 RCC-VE devices. Figure 9 presents throughput results and compares with ViT-Base as a baseline.

DeiT-Base, which has identical model structure as ViT-Base, achieves 0.62 images per second throughput on a single RCC-VE. With 4 devices, the DeiT-Base model achieves 0.95 images per second, outperforming ViT-Base's 0.82 images per second. DeiT-Small and DeiT-Tiny demonstrate more significant improvements – up to 5.55 and 17.23 images per second, respectively. As an orthogonal technique, model compression can potentially further improve PipeEdge's performance, making the combined approach a promising solution for large-scale model inference at the edge.

VI. Related Work

Current techniques to enable the execution of large models on edge devices mainly fall into two categories: single device optimization and distributed processing on multiple devices or servers. Aggressive model compression is an example of single device optimization methods. EdgeBERT [20] combines network pruning, entropy-based early exit, and adaptive attention span to reduce the model size and the inference latency of Bidirectional Encoder Representations from Transformers (BERT) for NLP tasks. Lite Transformer [52] adopts adaptive inference to reduce inference computation cost. Another promising solution includes neural architecture search (NAS), that trains a flexible supernet model to yield various subnets suitable for different targeted hardware platforms [22]. However, most of the above methods are either not suitable for distributed platforms or need redesigning and retraining of a pre-trained models and can potentially incur non-negligible drop in accuracy. In contrast, PipeEdge does not require retraining and does not reduce accuracy.

Through the assistance of cloud servers or distributed edge devices, the latency and computation for individual edge devices can be reduced without sacrificing accuracy. This strategy applies to both model training [53]–[55] and inference [56]–[58]. EdgePipe [59] leverages model parallelism and pipeline parallelism to accelerate model training over multiple edge devices. This work focuses on volatile wireless connections but only for homogeneous edge devices. Hermes [53] and Fjord [55] tackle the heterogeneity of data and devices for federated learning. They leverage data parallelism, so the model needs to be fit in the memory of each device. EDDL [54], an edge-based distributed deep learning system, addresses multiple challenges of performing training in realistic edge environments. Collaborative edge computing is also an effective technique for large-scale model inference and can be divided into two categories: cloud-based offloading and edge-based assistance. In several works [57], [58], [60]–[62], the distributed inference of DNN models on edge devices is partitioned and offloaded to cloud servers to reduce latency and minimize on-device computations. Considering the limited bandwidth and uncertain delay between the edge and the cloud, MoDNN [63] employs a MapReduce-like distributed inference paradigm and only utilizes idle mobile devices to execute CNN models. DeepThings [64] proposes a fine-grain partition method for CNN models on edge clusters. DeepHome [56] distributes machine learning inference tasks to multiple heterogeneous devices in the home. In [65], the proposed adaptive parallel inference method for CNN models is extended to heterogeneous edge devices.

With the emergence of transformer-based models, the model size continues to increase, making distributed execution more important. Megatron-LM [66] implements intra-layer partitioning for transformer-based models. This kind of parallelism has been applied in data centers with pipeline parallelism to train large-scale models [10], but this approach requires much higher bandwidth or devices with multiple accelerators, neither of which which are typical in edge environments. Pipeline parallelism has been proposed to address the problem of communication overheads. GPipe [25] presents effective pipeline parallelism for training large models on multiple TPU accelerators. PipeDream [29] and its subsequent work [30] target heterogeneous platforms and adopt pipeline parallelism to accelerate training. PipeMare [26] proposes a memory-efficient pipeline parallelism without sacrificing utilization. These works target data centers, and are difficult to directly apply to edge computing.

VII. Discussion

PipeEdge's main contribution is to accelerate large-scale neural network model inference using intelligent pipelining in heterogeneous distributed environments. The solution also then enables running larger—and more accurate—models than

single systems might otherwise support. This section discusses future research that is compatible with PipeEdge but beyond the scope of this paper.

The PipeEdge design generalizes beyond just edge systems and could also be suitable for cloud-only or edge-cloud collaborations. Heterogeneous cloud and data center-class systems can simply be treated as devices with strong computation capabilities by the partition scheduling algorithm. If there is a high-quality network between edge devices and cloud resources, the optimal partition schedule may lead to offloading some or all computation to cloud systems. PipeEdge will identify the optimal solution, regardless of whether work is run on a single edge device, pipelined between edge devices and/or cloud systems, or offloaded entirely to cloud systems.

Edge environments are typically more dynamic than data centers, e.g., devices come and go, resources may be shared, wireless network bandwidths and latencies vary, and devices are subject to changing power and energy constraints. PipeEdge's design is independent of these dynamics – adaptive runtime systems can respond to such dynamics by re-running PipeEdge's scheduling algorithm as their problem formulations dictate, or they can fine-tune systems and software after a schedule is deployed, e.g., to reduce power or energy in ways that (ideally) do not affect pipeline performance. Orthogonal machine learning techniques can also be applied to change model behaviors, e.g., using model compression to reduce computation (see Section V-E), or quantization to reduce data transmission sizes, both at the cost of reduced model accuracy. Finally, PipeEdge can be extended with other considerations, e.g., for geo-distributed computation where latency can vary and affect the system performance, Equation (1) could be amended to include a latency factor. We are exploring adapting to various runtime dynamics in future work.

VIII. Conclusion

This paper presents PipeEdge, a distributed inference approach for collaborative, heterogeneous, resource-constrained edge devices using pipeline parallelism. PipeEdge both improves throughput and enables larger models than individual memory-constrained devices can support on their own, without accuracy loss. We address the workload balance problem for heterogeneous clusters with an optimal dynamic programming-based partition method. We achieve $10.6\times$, $11.9\times$, and $12.78\times$ speedup with 16 devices for the ViT-Large, ViT-Huge, and BERT-Large models. PipeEdge demonstrates effectiveness and robustness for multiple heterogeneous cases, e.g., we show up to $4.16\times$ throughput speedup compared to GPipe and PipeDream when using a heterogeneous set of devices. Finally, we demonstrate that PipeEdge's throughput improvements are complementary to model compression by showing additional speedup with DeiT models.

Acknowledgments

This work was supported by the Department of the Navy, Office of Naval Research under, NSF, and DARPA with grant #N00014-20-1-2143, #1763747, and #HR00112190120, respectively. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies. This work was supported in parts by NSF and DARPA with grant numbers 1763747 and HR00112190120, respectively.

References

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

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[3] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," *arXiv preprint arXiv:2101.11986*, 2021.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[5] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision*. Springer, 2020, pp. 213–229.

[6] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[7] A. Y. Ding, E. Peltonen, T. Meuser, A. Aral, C. Becker, S. Dustdar, T. Hiessl, D. Kranzlmuller, M. Liyanage, S. Magshudi *et al.*, "Roadmap for edge ai: A dagstuhl perspective," *arXiv preprint arXiv:2112.00616*, 2021.

[8] J. Chen and X. Ran, "Deep learning with edge computing: A review." *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

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

[10] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. A. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on gpu clusters," 2021.

[11] A. Gholami, "AI and memory wall," Mar. 2021. [Online]. Available: https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8

[12] RCC-VE network board devices description. https://www.silicom-usa.com/pr/edge-networking-solutions/edge-network-boards/rcc-ve-network-board/.

[13] Minnowboard turbot devices description. https://www.silicom-usa.com/pr/eol/minnowboard-turbot/.

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

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

[16] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. Mahoney *et al.*, "Hawq-v3: Dyadic neural network quantization," in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 875–11 886.

[17] S. Kundu and S. Sundaresan, "Attentionlite: Towards efficient self-attention models for vision," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 2225–2229.

[18] S. Kundu, S. Wang, Q. Sun, P. A. Beerel, and M. Pedram, "Bmpq: bit-gradient sensitivity-driven mixed-precision quantization of dnns from scratch," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 588–591.

[19] S. Kundu, Y. Fu, B. Ye, P. A. Beerel, and M. Pedram, "Towards adversary aware non-iterative model pruning through d ynamic n etwork r ewiring of dnns," *ACM Transactions on Embedded Computing Systems (TECS)*, 2022.

[20] T. Tambe, C. Hooper, L. Pentecost, T. Jia, E.-Y. Yang, M. Donato, V. Sanh, P. Whatmough, A. M. Rush, D. Brooks *et al.*, "Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference," *arXiv preprint arXiv:2011.14203*, 2020.

[21] Q. Jin, L. Yang, and Z. Liao, "Adabits: Neural network quantization with adaptive bit-widths," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2146–2156.

[22] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "Hat: Hardware-aware transformers for efficient natural language processing," in *Annual Conference of the Association for Computational Linguistics*, 2020.

[23] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[24] S. Chen, Q. Li, M. Zhou, and A. Abusorrah, "Recent advances in collaborative scheduling of computing tasks in an edge computing paradigm," *Sensors*, vol. 21, no. 3, p. 779, 2021.

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

[26] B. Yang, J. Zhang, J. Li, C. Re, C. Aberger, and C. De Sa, "Pipemare: Asynchronous pipeline parallel dnn training," in *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica, Eds., vol. 3, 2021, pp. 269–296.

[27] Z. Li, S. Zhuang, S. Guo, D. Zhuo, H. Zhang, D. Song, and I. Stoica, "Terapipe: Token-level pipeline parallelism for training large-scale language models," *arXiv preprint arXiv:2102.07988*, 2021.

[28] C. He, S. Li, M. Soltanolkotabi, and S. Avestimehr, "Pipetransformer: Automated elastic pipelining for distributed training of transformers," *arXiv preprint arXiv:2102.03161*, 2021.

[29] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. Devanur, G. Granger, P. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *ACM Symposium on Operating Systems Principles (SOSP 2019)*, October 2019.

[30] D. Narayanan, A. Phanishayee, K. Shi, X. Chen, and M. Zaharia, "Memory-efficient pipeline-parallel dnn training," in *2021 International Conference on Machine Learning (ICML 2021)*, July 2021.

[31] J. H. Park, G. Yun, M. Y. Chang, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y.-r. Choi, "Hetpipe: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism," in *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 2020, pp. 307–321.

[32] C. Kim, H. Lee, M. Jeong, W. Baek, B. Yoon, I. Kim, S. Lim, and S. Kim, "torchgpipe: On-the-fly pipeline parallelism for training giant models," *CoRR*, vol. abs/2004.09910, 2020. [Online]. Available: https://arxiv.org/abs/2004.09910

[33] S. Kundu, Q. Sun, Y. Fu, M. Pedram, and P. Beerel, "Analyzing the confidentiality of undistillable teachers in knowledge distillation," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[34] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 184–199.

[35] S. Kundu, M. Nazemi, P. A. Beerel, and M. Pedram, "Dnr: A tunable robust pruning framework through dynamic network rewiring of dnns," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 344–350.

[36] T.-W. Chin, R. Ding, C. Zhang, and D. Marculescu, "Towards efficient model compression via learned global ranking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1518–1528.

[37] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Networks and Applications*, vol. 26, no. 3, pp. 1145–1168, 2021.

[38] H. Sharma, A. Haque, and F. Blaabjerg, "Machine learning in wireless sensor networks for smart cities: A survey," *Electronics*, vol. 10, no. 9, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/9/1012

[39] H. Isyanto, A. S. Arifin, and M. Suryanegara, "Design and implementation of iot-based smart home voice commands for disabled people using google assistant," in *2020 International Conference on Smart Technology and Applications (ICoSTA)*. IEEE, 2020, pp. 1–6.

[40] J. Cheng and T. Kunz, "A survey on smart home networking," *Carleton University, Systems and Computer Engineering, Technical Report SCE-09-10*, 2009.

[41] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

[42] S. d'Ascoli, H. Touvron, M. Leavitt, A. Morcos, G. Biroli, and L. Sagun, "Convit: Improving vision transformers with soft convolutional inductive biases," *arXiv preprint arXiv:2103.10697*, 2021.

[43] H. Su, V. Jampani, D. Sun, O. Gallo, E. Learned-Miller, and J. Kautz, "Pixel-adaptive convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 166–11 175.

[44] R. Goodfellow, S. Schwab, E. Kline, L. Thurlow, and G. Lawler, "The dcomp testbed," in *12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19)*. Santa Clara, CA: USENIX Association, Aug. 2019. [Online]. Available: https://www.mergetb.org/

[45] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the internet of things: A case study," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275–1284, 2018.

[46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[47] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online, Oct. 2020, pp. 38–45.

[48] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[49] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," 2019, in the Proceedings of ICLR.

[50] S. Kundu, M. Nazemi, M. Pedram, K. M. Chugg, and P. A. Beerel, "Pre-defined sparsity for low-complexity convolutional neural networks," *IEEE Transactions on Computers*, vol. 69, no. 7, pp. 1045–1058, 2020.

[51] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 347–10 357.

[52] Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han, "Lite transformer with long-short range attention," in *International Conference on Learning Representations*, 2019.

[53] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen, "Hermes: an efficient federated learning framework for heterogeneous mobile clients," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 420–437.

[54] P. Hao and Y. Zhang, "Eddl: A distributed deep learning system for resource-limited edge computing environment," in *Proceedings of the 6th ACM/IEEE Symposium on Edge Computing*, 2021.

[55] S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. I. Venieris, and N. D. Lane, "Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout," *arXiv preprint arXiv:2102.13451*, 2021.

[56] Z. Hu, A. B. Tarakji, V. Raheja, C. Phillips, T. Wang, and I. Mohomed, "Deephome: Distributed inference with heterogeneous devices in the edge," in *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, 2019, pp. 13–18.

[57] P. Ren, X. Qiao, Y. Huang, L. Liu, C. Pu, and S. Dustdar, "Fine-grained elastic partitioning for distributed dnn towards mobile web ar services in the 5g era," *IEEE Transactions on Services Computing*, 2021.

[58] K.-J. Hsu, K. Bhardwaj, and A. Gavrilovska, "Couper: Dnn model slicing for visual analytics containers at the edge," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 179–194.

[59] J. Yoon, Y. Byeon, J. Kim, and H. Lee, "Edgepipe: Tailoring pipeline parallelism with deep neural networks for volatile wireless edge devices," *IEEE Internet of Things Journal*, 2021.

[60] X. Chen, M. Li, H. Zhong, Y. Ma, and C.-H. Hsu, "Dnnoff: Offloading dnn-based intelligent iot applications in mobile edge computing," *IEEE Transactions on Industrial Informatics*, 2021.

[61] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, p. 615–629, Apr. 2017. [Online]. Available: https://doi.org/10.1145/3093337.3037698

[62] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2021.

[63] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1396–1401.

[64] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[65] L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, ser. SEC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 195–208. [Online]. Available: https://doi.org/10.1145/3318216.3363312

[66] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *CoRR*, vol. abs/1909.08053, 2019. [Online]. Available: http://arxiv.org/abs/1909.08053