Enabling Low Latency Edge Intelligence based on Multi-exit DNNs in the Wild

Zhaowu Huang[1†], Fang Dong[1†*], Dian Shen[1], Junxue Zhang[2], Huitian Wang[1], Guangxing Cai[1], Qiang He[3]
[1]*School of Computer Science and Engineering, Southeast University, Nanjing, China*
[2]*SING Lab, Hong Kong University of Science and Technology, Hong Kong, China*
[3]*School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia*
[†]*Zhaowu Huang and Fang Dong contributed equally to this work*
[*]*Fang Dong is the corresponding author of this paper*
Email:{zwh, fdong, dshen}@seu.edu.cn, jzhangcs@ust.hk, {huitwang, guangxingcai}@seu.edu.cn, qhe@swin.edu.au

Abstract*—In recent years, deep neural networks (DNNs) have witnessed a booming of artificial intelligence Internet of Things applications with stringent demands across high accuracy and low latency. A widely adopted solution is to process such computation-intensive DNNs inference tasks with edge computing. Nevertheless, existing edge-based DNN processing methods still cannot achieve acceptable performance due to the intensive transmission data and unnecessary computation. To address the above limitations, we take the advantage of Multi-exit DNNs (ME-DNNs) that allows the tasks to exit early at different depths of the DNN during inference, based on the input complexity. However, naively deploying ME-DNNs in edge still fails to deliver fast and consistent inference in the wild environment. Specifically, 1) at the model-level, unsuitable exit settings will increase additional computational overhead and will lead to excessive queuing delay; 2) at the computation-level, it is hard to sustain high performance consistently in the dynamic edge computing environment. In this paper, we present a Low Latency Edge Intelligence Scheme based on Multi-Exit DNNs (LEIME) to tackle the aforementioned problem. At the model-level, we propose an exit setting algorithm to automatically build optimal ME-DNNs with lower time complexity; At the computation-level, we present a distributed offloading mechanism to fine-tune the task dispatching at runtime to sustain high performance in the dynamic environment, which has the property of close-to-optimal performance guarantee. Finally, we implement a prototype system and extensively evaluate it through testbed and large-scale simulation experiments. Experimental results demonstrate that LEIME significantly improves applications' performance, achieving $1.1-18.7\times$ speedup in different situations.

***Index Terms*—Edge intelligence, Multi-exit setting, Computation offloading, DNN inference**

I. Introduction

With the rapid development of Internet of Things (IoT) end devices, an increasing number of deep neural networks (DNNs)-driven artificial intelligence IoT applications, spanning from autonomous driving [1] to face recognition [2], are emerging. These applications generate intensive data at the network edge and require mass computing resources to guarantee low response latency and high inference accuracy.

Currently, a widely adopted solution is to process such computation-intensive DNNs inference tasks with edge computing by providing a variety of resources to end devices in close proximity [3–7]. Such schemes treat the DNN as a computation directed acyclic graph (DAG) or a chain and partition it between end device and edge server. At run time, the end devices execute the part of the model and transmit the intermediate data, through the wireless network, to an edge server shared by end devices. The edge server continues the model execution and returns the results to end devices. However, the existing edge-based DNN processing methods still suffer from a set of limitations: the intensive intermediate data transferred to the edge incurs prohibitive network transmission latency; unnecessary computation further imposes a heavy burden on the edge, tremendously increasing the end-to-end response time.

To address the above limitations, we take the strength of Multi-exit DNNs (ME-DNNs) [8] which take the advantage of the fact that low complexity samples could be accurately classified in the earlier exit only with the low-level features that can be learned in the shallow layers of DNNs. Therefore, exit setting is to add some extra exits to the original DNNs to construct ME-DNNs. In this manner, each sample would ideally exit the network at the appropriate depth, saving inference time and computational resources. Besides, some tasks could early exit on the end device directly, avoiding the transmission overhead.

However, naively deploying ME-DNNs in edge still fails to deliver fast and consistent inference in the wild environment where the computational capabilities of nodes and network conditions have a tremendous difference, and task arrival rates vary dynamically. Specifically, we find the fact that: 1) at the model-level, unsuitable exit settings will increase additional computational overhead and will lead to excessive queuing delay. An improper exit setting leads to $4.47\times$ on average performance degradation (§II-B1); 2) at the computation-level, it is hard to sustain high performance consistently in the dynamic edge computing environment. In the wild edge, various changing factors, for example varying task arrival rates, mismatch with historical data during execution, leading to $2.85\times$ performance degradation on average (§II-B2).

In this paper, we present a Low Latency Edge Intelligence Scheme based on Multi-Exit DNNs (LEIME) to tackle the aforementioned problem. LEIME contains two components: at the model-level, we propose an exit setting algorithm to build an optimal ME-DNN with consideration of a vast range of factors in the wild edge; at the computation-level, we present an online distributed wild-edge-aware offloading mechanism to fine-tune the task dispatching at runtime to maintain high performance, responding to the transient mismatch between the tailored ME-DNN and the dynamic environment.

To find proper exits to build a tailored ME-DNN efficiently, we use the branch and bound method to narrow the searching space, which can obtain the optimal exit setting with $O(mln(m))$ time complexity. To ensure that LEIME can provide consistent low latency in the wild edge, we model the offloading problem as a stochastic optimization problem to minimize the average task completion time (TCT) in the long term, which balancing the loads of end devices and edge server. Based on Lyapunov optimization [9], the long-term stochastic optimization problem is converted to a deterministic optimization problem. A distributed fine-grained offloading algorithm is introduced to solve it with provable close-to-optimal performance.

To evaluate LEIME's performance, we implement a prototype system and conduct extensive testbed and simulation experiments. Evaluation results show that LEIME can significantly improve application's latency, achieving $1.1 - 18.7\times$ speedup under different situations. Moreover, because of LEIME's awareness of the changing environment, it remains high performance consistently.

The contributions of this paper are summarized as follows:

- We comprehensively analyze and identify the key factors causing performance degradation when deploying the ME-DNNs in the edge, including model-level and computation-level.
- At the model-level, we propose an exit setting algorithm to automatically build the optimal ME-DNNs with lower time complexity.
- At the computation-level, we present a distributed offloading mechanism to fine-tune the task dispatching at runtime to maintain high performance in dynamic environments, which has the property of close-to-optimal performance guarantee.
- We implement and evaluate our method through large scale testbed and simulation experiments. The experimental results show that the LEIME achieves lower latecny and remains high performance consistently, compared with the state-of-the-art methods.

II. Background and Motivation

A. Wild Edge Environment

Different from traditional cloud computing, edge intelligence tries to elaborate devices, edge, and cloud simultaneously to deliver low latency applications. The heterogeneous architecture of edge intelligence leads to a wild environment filled with diversities and uncertainties, as summarized below:

Edge and end devices share varying computation capabilities: There are various edge servers, e.g., a gateway in a smart home, a desktop computer, a micro data center [10][11], whose computational capability varies a lot. For example,

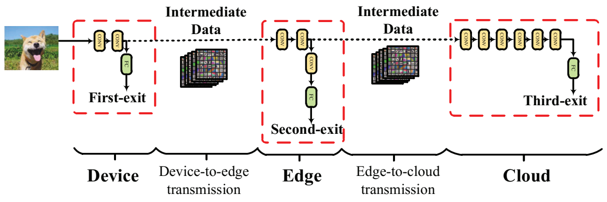


Fig. 1. Co-inference scheme on device-edge-cloud architecture

an edge server desktop computer with a GPU (NVIDIA Geforce940MX) outperforms a laptop computer with an Intel i5-7200U CPU by $5\times$ while performing ResNet50 [12] inference. Devices connected to the same edge are also heterogeneous, and their computational capabilities are quite different, e.g., Raspberry Pi 3B+ with ARM Cortex-A53 CPU and Jetson Nano with NVIDIA Maxwell GPU, have $8.2\times$ performance difference in Inception V3 [13].

Network condition varies dramatically: The network environment between devices and the edge servers is heterogeneous, such as bandwidth and propagation delay of the link connecting end devices and edge, e.g., the bandwidth and propagation delay vary from 1-30Mbps and 10-200ms.

Applications have different characteristics: The intelligence applications running on end devices have different characteristics, including the complexity of the input data, the task arrival rate, DNNs structure, and deadline requirements.

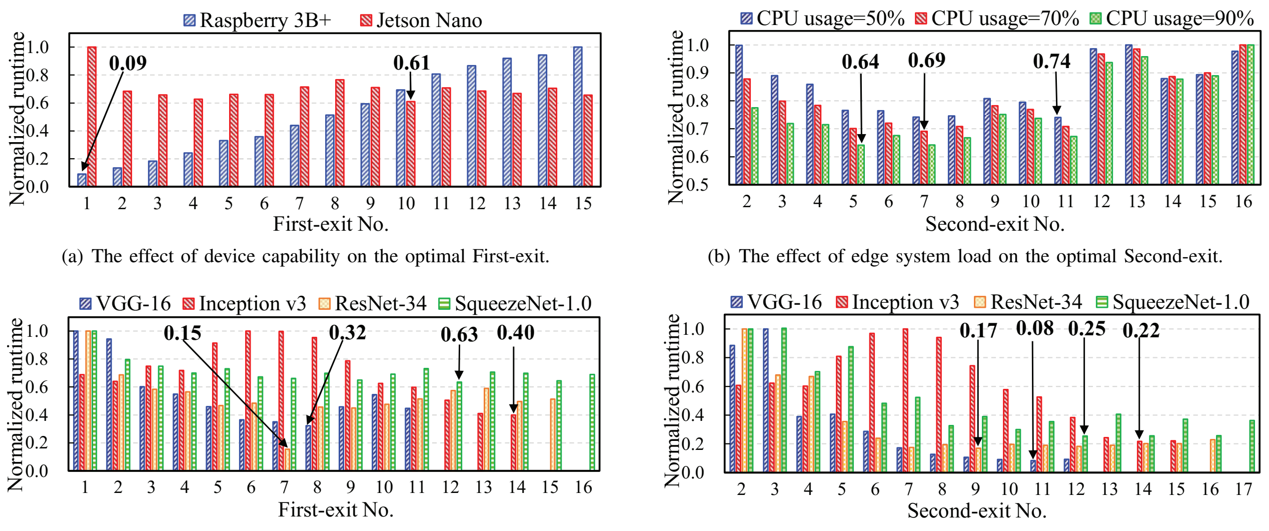
The above factors contribute to the wild edge, posing dramatic challenges to deploy ME-DNNs in the edge. The exit-setting is difficult towards a vast range of historical factors, such as heterogeneous device computation capabilities. Moreover, the changing online factors, such as arrival rate, may cause a transient mismatch with historical data, causing degraded performance in the long term. We will analyze the above 2 problems in the following sections.

B. Multi-exit DNNs in the Edge

To analyze the performance of ME-DNNs in the edge, we setup a real-world testbed shown in Fig. 1. CIFAR-10 is used as our experiment dataset. We build a ME-DNN with 3 exits, First-exit, Second-exit, and Third-exit. Therefore, the ME-DNN is partitioned into 3 blocks, which will be deployed on the end device, edge, and cloud, respectively. All inference tasks are launched from end devices. If the task does not exit in advance on First-exit, due to low confidence, the intermediate data will be transferred to the edge to continue the inference. Similarly, the tasks that cannot exit on the Second-exit will be performed further on the cloud.

1) Non-optimal exit setting causes degraded performance: To comprehensively investigate the impact of exit setting towards performance, we change the First-exit and Second-exit respectively to measure the average inference latency of 1000 recognition tasks towards different images.

Exit settings under different computational capabilities of nodes In Fig. 2(a) and (b), we present the exit settings results under different computational capabilities of nodes. As shown in Fig. 2(a), with Raspberry pi 3B+ [14], the optimum First-exit is achieved at $exit$-1, which involves the



(a) The effect of device capability on the optimal First-exit.

(b) The effect of edge system load on the optimal Second-exit.

(c) The effect of DNN type on the optimal First-exit.

(d) The effect of DNN type on the optimal Second-exit.

Fig. 2. The effect of system computing capability, and DNN type on the optimal exit settings. (a)(b) illustrate the optimal exit settings under under different computational capabilities of nodes, and (c)(d) illustrate optimal exit settings under different DNNs. The arrows in the figures indicate the optimal exit in each case and the corresponding normalized latency, e.g., (a) indicates that the optimal First-exit is $exit$-1 on Raspberry pi and $exit$-10 on Jetson Nano.

least computation. On the contrary, with Jetson Nano [15], that is $> 10\times$ faster than Raspberry pi, the optimum First-exit is achieved at $exit$-10, which tries to reduce the transmission of intermediate data. Fig. 2(b) indicate that the optimal solution is to choose a deeper Second-exit to saturate the computation capacity of the server, when the computing load of the edge server is light. Otherwise, we choose a more shallow Second-exit for a heavily loaded server to reach optimal.

Exit settings under different DNNs In Fig. 2(c) and (d), we study optimal First-exit and Second-exit settings under different DNNs respectively. The results indicate that under different DNNs (VGG-16, ResNet-34, Inception v3, and SqueezeNet 1.0) optimum exit settings are different. This is because the structure for different DNNs, including the intermediate data size of each layer and the amount of calculation of each layer, are different, which will affect the computation and transmission latency for inferring tasks.

Conclusion: It is not easy to optimally setup exits for a ME-DNN to deliver low latency applications. An improper exit setting causes $4.47\times$ on average performance degradation. The optimal solution is related to many factors, such as device computational capacities, model features, each with a different strategy to achieve optimum. Furthermore, in the wild edge, these factors do not exist solely, leading to further challenges in setting up the proper exits for a ME-DNN.

2) Non-optimal task offloading further degrades the performance: In the wild edge, besides the above-mentioned static factors that can be known in advance, there are many dynamic factors, e.g., task arrival rates and networks. The current practice is to use task offloading strategies to handle dynamic factors [16]. We are trying to investigate how dynamic factors affects application performance under different offloading strategies. In the following experiments, we use the trained Multi-exit Inception v3 [13] whose First-exit, Second-exit, and Third-exit are fixed at $exit$-1, 14, and 16.

Varying task arrival rate As shown in Fig. 3(a), we can find that under different task arrival rates, the optimal task offloading ratio (the percentage of tasks launched from the edge server) to achieve lowest latency varies.

Varying data complexity In this experiment, we synthesize some different datasets to build different complexity datasets that are reflected by the exit rate (i.e. exit probability) of First-exit. In Fig. 3(b), when we have varying data complexity, the optimal offloading strategies also vary.

Varying networking conditions Fig. 3(c) and (d) show the results of our third experiments where we change the bandwidth and propagation delay. Similarly, the optimal offloading strategy varies when we have different networking conditions.

Conclusion: From the above experiments, we observe that those dynamic factors lead to dynamic optimal offloading strategies. An improper task offloading strategy causes $2.85\times$ on average performance degradation. Furthermore, in the wild edge, those factors may change dramatically and unpredictably, leading to increasing difficulties to leverage an existing offloading strategy, causing suboptimal performance.

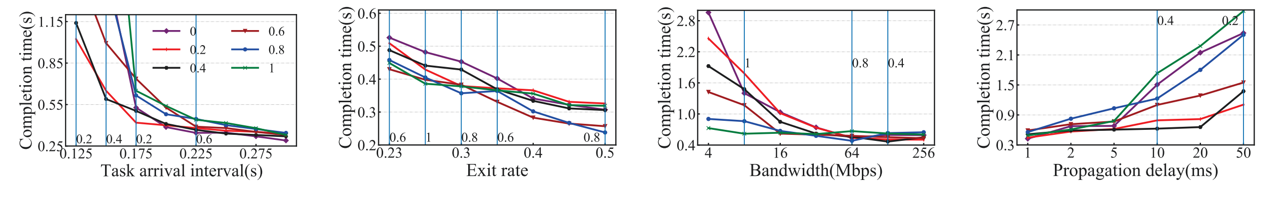
III. Low Latency Edge Intelligence Scheme based on Multi-Exit DNNs

To achieve consistent low latency DNN inference in the wild edge, a Low Latency Edge Intelligence Scheme based on Multi-Exit DNNs (LEIME) is proposed to **minimize the average task completion time (TCT)** in the long term.

A. Overview of LEIME

As shown in Fig. 4, LEIME contains two components:

1) Exit Setting: In Fig. 4 (upper), LEIME adds two extra exits to the original DNN to build a ME-DNN which contains First-exit (Exit 1), Second-exit (Exit 2), and Third-exit (Exit



(a) The effect of task arrival interval. (b) The effect of First-exit exit rate. (c) The effect of bandwidth. (d) The effect of propagation delay.

Fig. 3. The curves show the TCT under different task offloading ratios (from 0 to 1). E.g., the black line represents that the number of tasks offloaded to the edge starts to execute, accounting for 0.4. The blue vertical line in the figure represents the optimal offloading ratio in the current abscissa. E.g., as shown in (c), when the bandwidth is 8 Mbps, the optimal offloading ratio is 1, when the bandwidth increases to 128 Mbps, the optimum offloading ratio becomes 0.4.

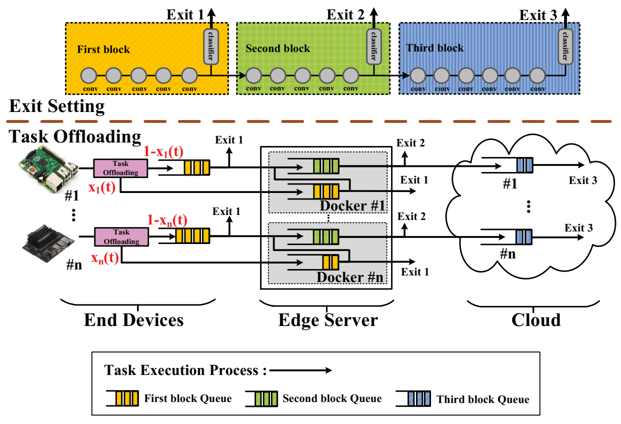


Fig. 4. Illustrating the overview of LEIME which contains two major components: the exit setting and partition of the model (upper); the offloading and execution process of the inference task (lower).

3). In LEIME, according to the exit positions, ME-DNN is partitioned into three blocks, including the first block, second block, and third block, which are deployed into end devices, edge server, and cloud.

2) Task Offloading: In LEIME, tasks generated on devices are sent to the edge server or left in end devices to launch the inference process, based on the task offloading algorithm. Fig. 4 (lower) shows the task will be further inferred by the second block of ME-DNN deployed on the edge server if the inference result of the First-exit cannot meet the confidence threshold. In the worst case, the input data with high complexity, the intermediate data will be sent to the cloud for the final inference of the third block.

B. System model

1) Edge computing model: We consider an ME-DNN based edge computing system that consists of a set $\mathcal{N} = \{1, 2, ..., N\}$ of end devices, an edge server, and a remote cloud, as illustrated in Fig. 4(lower). The floating point operations per second (FLOPS) of i-th end device, edge server, and cloud are denoted as F_i^d, F^e, and F^c respectively. Let F_{av}^d and F_{av}^e denote the average available device and edge server FLOPS capability of devices based on the historical statistics data. We consider that the computing resources allocated to the i-th end device by the edge is denoted as $p_i, i \in \mathcal{N}$, with constraints $0 < p_i < 1$ and $\sum_{i\in\mathcal{N}} p_i = 1$. The derivation process p_i can be obtained in Appendix B. Let B_i^e and L_i^e denote the bandwidth and latency between the i-th end device and edge. Similarly, let B_{av}^e and L_{av}^e denote the average available bandwidth and latency (the connection latency caused by establishing the protocols) between the devices and edge server. Let B_{av}^c and L_{av}^c denote the average available bandwidth and latency between the edge server and cloud. We divide time into discrete time slots and denote different time slots with $T = \{1, 2, 3, ...\}$. The time slot length is τ. Let us denote the number of tasks of the i-th device in the t-th time slot as $M_i(t)$ which is independent and identically distributed in different time slot within $[0, M_{i,max}], M_{i,max} \in \mathbb{R}^+$, with the expectation $\mathbb{E}[M_i(t)] = k_i, k_i \in [0, M_{i,max}], i \in \mathcal{N}$. In the t-th time slot, the task offloading ratio of the i-th end device is $x_i(t)$, which means that tasks launched locally account for $1 - x_i(t)$. The range of $x_i(t)$ is $0 \leq x_i(t) \leq 1, \forall i \in \mathcal{N}$.

2) Task model: In this paper, we consider convolutional layers (hereafter called layers) as the atomic elements of a DNN model because the convolutional layer occupies the most floating point operations (FLOPs) in DNNs, same as the state-of-the-art works [6, 7]. We model a DNN network as a chain structure with a set $\mathbb{M} = \{l_1, l_2, ..., l_m\}$ of layers. Let μ_{l_i} and d_{l_i} denote FLOPs and the amount of intermediate data of the i-th layer of DNN. In ME-DNNs, at most one exit that is essentially a classifier composed of a pooling layer, two fully connected layers, and a softmax layer will be set between adjacent convolutional layers. Therefore, m classifiers are added between adjacent network layers of a DNN to construct m candidate exits $EXIT = \{exit_1, exit_2, ..., exit_m\}$. Let μ_{exit_i} represents the FLOPs of the i-th exit. At each exit, we use the maximum value of the softmax layer output (called confidence) to determine whether a task can exit early. In detail, a confidence threshold is set at each exit. Only the confidence of tasks higher than the threshold, the tasks can exit inference early. The higher the threshold, the higher the accuracy of ME-DNNs, but the more difficult tasks to exit early. Therefore, we strictly set the threshold of each exit to make the task can exit early efficiently while guaranteeing inference accuracy. In this way, we can obtain the exit rate (i.e. exit probability) $\{\sigma_{exit_1}, \sigma_{exit_2}, ..., \sigma_{exit_m}\}$ based on the aboved threshold, where $\sigma_{exit_m} = 100\%$. The notations and definitions used in the model are summarized in Table I.

C. Multi-exit DNN Design

Let $\{\frac{\mu_{l_i}}{F_{av}^d}, \frac{\mu_{l_i}}{F_{av}^e}, \frac{\mu_{l_i}}{F^c}\}$ and $\{\frac{\mu_{exit_i}}{F_{av}^d}, \frac{\mu_{exit_i}}{F_{av}^e}, \frac{\mu_{exit_i}}{F^c}\}$ denote the computing time of the i-th layer and the i-th exit classifier at the device, edge, and cloud based on the above model. $\frac{d_{l_i}}{B_{av}^e}$

TABLE I
KEY NOTATIONS

Notation	Definition
$\sigma_1, \sigma_2, \sigma_3$	The exit probability of the three exits
μ_1, μ_2, μ_3	The FLOPs required of the three blocks of ME-DNN
d_0, d_1, d_2	The data size of the input, First-, and Second-exit
F_i^d, F^e, F^c	The FLOPS of the i-th device, edge server, and cloud
F_{av}^d, F_{av}^e	The average available FLOPS of devices and edge server
B_i^e, L_i^e	The bandwidth and latency between device i and edge
B_{av}^e, L_{av}^e	The average available bandwidth and latency of devices
B_{av}^c, L_{av}^c	The average available bandwidth and latency of edge
μ_{l_i}, μ_{exit_i}	The FLOPs of the i-th layer and exit
d_{l_i}	The amount of intermediate data of the i-th layer
p_i	Resource allocation ratio of i-th device
k_i	The number of tasks expectation of the i-th device
$x_i(t)$	The offloading ratio of the i-th device in t-th time slot
τ	The length of time slot

and $\frac{d_{l_i}}{B_{av}^c}$ denote the device-to-edge transmission delay and the edge-to-cloud transmission delay.

In LEIME, 3 exits are selected from m candidate ones to build a ME-DNN which will be partitioned into three blocks. Specially, we take the original exit (i.e. $exit_m$) of DNNs as the Third-exit. We consider any First-Second-Third exit combination $E = \{e_1, e_2, e_3\}$, where $e_3 = exit_m$.

The time cost on the end device consists of the computing time of layers (from layer 1 to layer r_1) and the First-exit (i.e. classifier) e_1, as expressed (1).

$$t^d = \sum_{j=1}^{r_1} \frac{\mu_{l_j}}{F_{av}^d} + \frac{\mu_{e_1}}{F_{av}^d} \tag{1}$$

The time cost on the edge server consists of the computing time of layers (from layer r_1+1 to layer r_2) and the Second-exit (i.e. classifier) e_2, the transmission time of intermediate data from end device to edge server, including the connection latency, as expressed (2).

$$t^e = \sum_{j=r_1+1}^{r_2} \frac{\mu_{l_j}}{F_{av}^e} + \frac{\mu_{e_2}}{F_{av}^e} + \frac{d_{r_1}}{B_{av}^e} + L_{av}^e \tag{2}$$

Similarly, the time cost on the cloud consists of the computing time of layers (from layer r_2+1 to layer r_3) and the Third-exit e_3, the transmission time of intermediate data from edge server to cloud, including the connection latency, as expressed (3).

$$t^c = \sum_{j=r_2+1}^{m} \frac{\mu_{l_j}}{F^c} + \frac{\mu_{e_3}}{F^c} + \frac{d_{r_2}}{B_{av}^c} + L_{av}^c \tag{3}$$

Given tasks could be completed with a certain probability at every exit, saving a part of time, the total time cost $T(E)$ for exit combination E consist of t^d, t^e, and t^c except the part of early exit tasks. Therefore, the objective of LEIME's exit setting is minimization $T(E)$, which can be described as a optimization problem $P0$:

$$\begin{aligned} P0: min \quad & T(E) = \sigma_{e_3}(t^d + t^e + t^c) - (\sigma_{e_1} t^e + \sigma_{e_2} t^c) \\ s.t. \quad & e_1, e_2, e_3 \in \{exit_1, exit_2, ..., exit_m\} \\ & e_1 < e_2 < e_3 \end{aligned} \tag{4}$$

where $\sigma_{e_1} t^e + \sigma_{e_2} t^c$ denotes the saving time of early exit tasks.

We can find that the problem $P0$ is an integer optimization problem (IP). We propose a branch-and-bound algorithm based on Theorem 1 to narrow the search space.

Theorem 1. *First, we consider a ME-DNN with 2 exits $\{exit_1, exit_m, -\}$.The ME-DNN is partitioned into 2 blocks, one is processed in end device and another in edge server. If $T(\{exit_{i_1}, exit_m, -\}) \le T(\{exit_{i_2}, exit_m, -\})$ & $exit_{i_1} < exit_{i_2}$, $\forall exit_{i_1}$, $exit_{i_2} \in EXIT$. Then $T(E_1) < T(E_2)$ hold, where $E_1 = \{exit_{i_1}, exit_j, exit_m\}, E_2 = \{exit_{i_2}, exit_j, exit_m\}$, $\forall j, exit_{i_{1,2}} < exit_j < exit_m$.*

Proof:

$$\begin{aligned} T(\{exit_i, exit_m, -\}) &= \sigma_{exit_m}(\sum_{j=1}^{i} \frac{\mu_{l_j}}{F_{av}^d} + \sum_{j=i+1}^{m} \frac{\mu_{l_j}}{F_{av}^e} \\ &+ \frac{\mu_{exit_i}}{F_{av}^d} + \frac{\mu_{exit_m}}{F_{av}^e} + \frac{d_{l_i}}{B_{av}^e} + L_{av}^e) - \sigma_{exit_i}(\sum_{j=i+1}^{m} \frac{\mu_{l_j}}{F_{av}^e} \\ &+ \frac{\mu_{exit_m}}{F_{av}^e} + \frac{d_{l_i}}{B_{av}^e} + L_{av}^e) \end{aligned} \tag{5}$$

We take Third-exit into consideration, and construct two exit combinations $E_1 = \{exit_{i_1}, exit_j, exit_m\}, E_2 = \{exit_{i_2}, exit_j, exit_m\}$. The corresponding time cost is $T(E_1), T(E_2)$, we can get:

$$\begin{aligned} & T(E_1) - T(E_2) = \\ & T(\{exit_{i_1}, exit_m, -\}) - T(\{exit_{i_2}, exit_m, -\}) \\ & + (\sigma_{i_1} - \sigma_{i_2})(\sum_{k=j+1}^{m} \frac{\mu_{l_k}}{F_{av}^e} + \frac{\mu_{exit_j}}{F_{av}^e} - \frac{\mu_{exit_m}}{F_{av}^e}) \end{aligned} \tag{6}$$

Considering a general situation, the deeper the exit location, the higher the exit probability σ. Therefore, if the depth of $exit_{i_1}$ is shallower than $exit_{i_2}$, then $\sigma_{i_1} - \sigma_{i_2} < 0$. And obviously $(\sum_{k=j+1}^{m} \frac{\mu_{l_k}}{F^e} + \frac{\mu_{exit_j}}{F_{av}^e} - \frac{\mu_{exit_m}}{F_{av}^e}) > 0$, therefore if $T(\{exit_{i_1}, exit_m, -\}) < T(\{exit_{i_2}, exit_m, -\})$, we can guarantee $T(E_1) - T(E_2) < 0$. ■

According to the Theorem 1, we can narrow the search space of the First-exit to narrow the overall search space. If $exit_{i_k} = \arg\min_{exit_i}\{T(\{exit_i, e_m, -\})\}$, then the First-exit must $\le exit_{i_k}$. Firstly, we initialize the search upper bound (*upbound*) of the First-exit to $m-2$. In each round of search, take the current optimal device exit as $exit_{i_k} = \arg\min_{exit_i}\{T(\{exit_i, e_m, -\}) \mid 1 \le i \le upbound\}$, where k is the number of round. The optimal exit settings of each round only needs to be searched in the combination set R_{i_k} of $exit_{i_k}, exit_j (i_k < j < m-1), exit_m$, and exclude the possibility of $exit_i (i_k < i < i_{k-1})$ as the of device exit. Then update $upbound = i_k - 1$ with the selection results of the First-exits in each round. Each round of search keeps narrowing the search space until $upbound = 0$. When the search ends, an optimal exit setting for ME-DNN is as follows:

$$E_{best} = \arg\min_{E}\{T(E) \mid E \in R_{i_1} \cup R_{i_2} \cup ... \cup R_{i_k}\} \tag{7}$$

Based on the proposed algorithm, we can get the optimal exit setting. In the best case, time complexity is $O(m)$.

Because we only need to traverse the $m-2$ exits to select the Second-exit if the First-exit is $exit_1$. The average complexity can be seen in Theorem 2.

Theorem 2. *The average time complexity of exit setting algorithm is $O(mln(m))$.*

Proof: See Appendix A. ■

Based on exit setting algorithm, we can adaptively select three exits to build the most proper ME-DNN in the wild edge. The exit probability of the three exits are denoted as $[\sigma_1, \sigma_2, \sigma_3]$ respectively. According to the exit setting, the ME-DNN can be divided into 3 blocks which require $[\mu_1, \mu_2, \mu_3]$ FLOPs. The size of the raw input, the intermediate data of the First-exit and Second-exit are denoted as $[d_0, d_1, d_2]$.

D. Task offloading model and solution

1) Task Offloading And Task Execution: It is noteworthy to mention that the inference tasks of the second and third blocks are processed fixedly on edge and cloud respectively. Therefore, we only need to focus on the process of first block inference in LEIME. For the i-th end device, $A_i(t) = (1 - x_i(t))k_i$ and $D_i(t) = x_i(t)k_i$ denote the numbers of first block inference tasks on the device and edge server in the t-th time slot. Therefore, the amount of data transferred from the end device to the edge server consists of the raw input data and the intermediate data of First-exit if the task cannot exit early in it. It is limited by the bandwidth and latency between end devices and the edge, which is formulated as (8).

$$D_i(t)d_0 + A_i(t)(1-\sigma_1)d_1 \le B_i^e(\tau - L_i^e) \tag{8}$$

For the i-th end device, the $D_i(t)$ first block inference tasks and all the second block inference tasks are processed on the edge using p_iF^e computing resources. Thus, the computing resources used to process the first and second blocks inference tasks on the edge server meet equation $\frac{F_{i,1}^e}{F_{i,2}^e} = \frac{x_i(t)\mu_1}{(1-\sigma_1)\mu_2}, \forall i \in \mathcal{N}$. Therefore, The $F_{i,1}^e$ can be derived as:

$$F_{i,1}^e = \frac{x_i(t)\mu_1 p_i(t)F^e}{x_i(t)\mu_1 + (1-\sigma_1)\mu_2}, \forall i \in \mathcal{N} \tag{9}$$

The number of first block inference tasks of i-th end device that processed by local and edge within each time slot are $b_i(t) = \frac{F_i^l\tau}{\mu_1}, \forall i \in \mathcal{N}$ and $c_i(t) = \frac{F_{i,1}^e\tau}{\mu_1}, \forall i \in \mathcal{N}$.

As shown in Fig.4, at the beginning of the t-th time slot, the local queue length of the i-th end device is $Q_i(t)$. In the t-th time slot, the tasks which has arrived but not been processed will be put into $Q_i(t)$. Future states are driven by stochastic arrival $A_i(t)$ and server process $b_i(t)$ according to:

$$Q_i(t+1) = max\{Q_i(t) - b_i(t), 0\} + A_i(t), \forall i \in \mathcal{N} \tag{10}$$

Similarly, task queue $H_i(t)$ denote the number of first block inference tasks of the i-th device which has arrived but not been executed in the edge, which can be derived according to:

$$H_i(t+1) = max\{H_i(t) - c_i(t), 0\} + D_i(t), \forall i \in \mathcal{N} \tag{11}$$

2) Computation and Transmission Delay Cost: To simplify our analysis, we assume that tasks are generated at the beginning of each time slot, and performed in a (first-in-first-out) FIFO manner. In each time slot, the arriving task will be divided into two parts which will be processed from the end device or edge server. Therefore, we will analyze the computation and transmission delay cost of the end device and edge server.

End device: For the tasks arriving in the t-th time slot and launched on the end device, it needs to wait for the tasks in the $Q_i(t), \forall i \in \mathcal{N}$ to be emptied before processed. Thus, the waiting time is expressed as $C_{i,1}^d(t) = A_i(t)\frac{Q_i(t)\mu_1}{F_i^d}, \forall i \in \mathcal{N}$. For the tasks arriving in t-th time slot, the sum of their processing time and queuing time is expressed as $C_{i,2}^d(t) = A_i(t)\frac{\mu_1}{F_i^d} + \frac{A_i(t)\times(A_i(t)-1)}{2}\frac{\mu_1}{F_i^d}, \forall i \in \mathcal{N}$. If a task can not be finished in the First-exit, the intermediate data needs to be sent to the edge server for further inference. The transmission time of intermediate data can be expressed as $C_{i,3}^d(t) = (1-\sigma_1)A_i(t)(\frac{d_1}{B_i^e} + L_i^e), \forall i \in \mathcal{N}$. Therefore, the total time cost of tasks that arriving in t-th time slot and processed locally is the sum of $C_{i,1}^d(t)$, $C_{i,2}^d(t)$, and $C_{i,3}^d(t)$, which denoted as $T_i^d(t)$.

$$T_i^d(t) = C_{i,1}^d(t) + C_{i,2}^d(t) + C_{i,3}^d(t), \forall i \in \mathcal{N} \tag{12}$$

Edge server: In the t-th time slot, $D_i(t)$ tasks are offloaded into edge server by the i-th end device. Thus, the input data should be transferred into the edge server and its time cost is expressed as $C_{i,1}^e(t) = D_i(t)(\frac{d_0}{B_i^e} + L_i^e)$. Similarly, the tasks transmitted to the edge server need to wait for the tasks in queue $H_i(t)$ to be emptied, and the waiting time can be expressed as $C_{i,2}^e(t) = D_i(t)\frac{H_i(t)\mu_1}{F_{i,1}^e}$. And the sum of the processing time and queuing time of tasks that arriving in the t-th time slot is expressed as $C_{i,3}^e(t) = D_i(t)\frac{\mu_1}{F_{i,1}^e} + \frac{D_i(t)\times(D_i(t)-1)}{2}\frac{\mu_1}{F_{i,1}^e}$. Therefore, the total time cost of tasks that arriving in the t-th time slot and performed on the edge is the sum of $C_{i,1}^e(t)$, $C_{i,2}^e(t)$, and $C_{i,3}^e(t)$, which denoted as $T_i^e(t)$.

$$T_i^e(t) = C_{i,1}^e(t) + C_{i,2}^e(t) + C_{i,3}^e(t), \forall i \in \mathcal{N} \tag{13}$$

Therefore, for the tasks arriving at the i-th end device in the t-th time slot, the total time cost of end devices and edge server expectation is $Y_i(t)$.

$$Y_i(t) = T_i^d(t) + T_i^e(t), \forall i \in \mathcal{N} \tag{14}$$

3) Problem Formulation: The goal of LEIME is to minimize average TCT in the long term:

$$\begin{aligned}
P1: &\min_{\{X(t)\}} \lim_{T\to+\infty} \frac{1}{T}\sum_{T-1}^{0} E\left[\sum_{i\in N} Y_i(t)\right] \\
s.t. \quad & C1: 0 \le x_i(t) \le 1, \forall i \in \mathcal{N}, t \in T \\
& C2: (8), t \in T \\
& C3: \lim_{T\to+\infty} \frac{E[|Q_i(T)|]}{T} = 0, \forall i \in \mathcal{N} \\
& C4: \lim_{T\to+\infty} \frac{E[|H_i(T)|]}{T} = 0, \forall i \in \mathcal{N}
\end{aligned} \tag{15}$$

where $P \triangleq \{p_i | i \in \mathcal{N}\}$ and $X(t) \triangleq \{x_i(t) | i \in \mathcal{N}\}$. In $P1$, $C3$ and $C4$ guarantee that the queues $Q_i(t)$ and $H_i(t)$ are mean rate stable in a discrete time process [9].

In the wild edge, the environment fluctuates dynamically in the short term, but it can be considered stable from a long-term perspective. Therefore, based on Lyapunov optimization [9], $P1$ can be converted to a deterministic optimization problem within each time slot, while the prior system information and task arrival distribution are not required. Let $\mathbf{\Theta}(t) = [Q(t), H(t)]$ be the queue backlog queue, where $Q(t) = (Q_1(t), Q_2(t), ..., Q_n(t))$, $H(t) = (H_1(t), H_2(t), ..., H_n(t))$. To model the problem as a Lyapunov optimization problem, we define the quadratic Lyapunov function as $L(\mathbf{\Theta}(t)) = \frac{1}{2}\sum_{i\in N}\left[Q_i^2(t) + H_i^2(t)\right]$. Then, the condition Lyapunov drift is defined as $\Delta(\mathbf{\Theta}(t)) = \mathbb{E}\left[(L(\mathbf{\Theta}(t+1)) - L(\mathbf{\Theta}(t)))|\mathbf{\Theta}(t)\right]$.

The underlying objective of our optimal control decision is to minimize a supremum bound on the following drift-plus-penalty expression in each time slot.

$$\Delta(\mathbf{\Theta}(t)) + V \times \mathbb{E}[Y(t)|\mathbf{\Theta}(t)] \tag{16}$$

where $Y(t) = \sum_{i\in N} Y_i(t)$ and V is a non-negative control parameter that is chosen as desired [9].

Lemma 1. *According to the theory of and Lyapunov optimization [9], under any control algorithm, the drift-plus-penalty expression has the following upper bound for all t, all possible values of $\mathbf{\Theta}(t)$, and all parameters $V \geq 0$:*

$$\begin{aligned}&(16) \leq B + V \times E[Y(t)|\mathbf{\Theta}(t)] + E[\sum_{i\in N} Q_i(t)(A_i(t)- \\ &b_i(t))|\mathbf{\Theta}(t)] + E[\sum_{i\in N} H_i(t)(D_i(t) - c_i(t))|\mathbf{\Theta}(t)]\end{aligned} \tag{17}$$

where B is a positive constant [9].

Proof: See Appendix C. ■

According to Lemma 1, to minimize $Y(t)$ and maintain the stability of task queues, we need to minimize the upper bound of the drift-plus-penalty function in each time slot as expressed in formula (17). Since B is a constant, the problem $P1$ is transformed as:

$$\begin{aligned}P1' : \underset{\{X(t)\}}{min} &\sum_{i\in N} V \cdot Y_i(t) + Q_i(t)(A_i(t) - b_i(t)) \\ &+ H_i(t)(D_i(t) - c_i(t)) \\ &s.t. 0 \leq x_i(t) \leq 1, \forall i \in \mathcal{N}, t \in T\end{aligned} \tag{18}$$

4) Decentralized Optimization for P1': $P1'$ can be proved to be a convex problem, which can be solved by some common methods, such as gradient descent method, quasi-Newton method [17]. However, such centralized solutions are time-consuming in the case of large-scale end device connections. Therefore, we propose a decentralized optimization algorithm to minimize the average TCT.

Let us substitute (14) into (18), and we can rewrite the objective function of each end device as (19).

$$\begin{aligned}&VT_i^d(t) + Q_i(t)(A_i(t) - b_i(t)) + VT_i^e(t) + \\ &H_i(t)(D_i(t) - c_i(t))\end{aligned} \tag{19}$$

We divide formula (19) into two parts which are defined as $Y_i^d(t) = VT_i^d(t) + Q_i(t)(A_i(t) - b_i(t))$ and $Y_i^e(t) = VT_i^e(t) + H_i(t)(D_i(t) - c_i(t))$. We can now use the Cauchy-Schwarz inequality to obtain the following inequality.

$$Y_i^d(t) + Y_i^e(t) \geq 2\sqrt{Y_i^d(t)Y_i^e(t)} \tag{20}$$

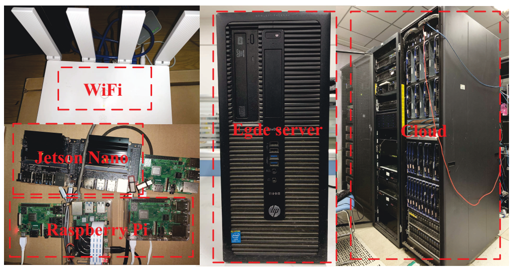


Fig. 5. Experiment system

If and only if $Y_i^d(t) = Y_i^e(t)$, the inequality (20) takes the equal sign. Divide both sides of the equation by V, we can get $\frac{Y_i^d(t)}{V} = \frac{Y_i^e(t)}{V}$. Since V can take $+\infty$, $\frac{Q_i(t)(A_i(t)-b_i(t))}{V}$ and $\frac{H_i(t)(D_i(t)-c_i(t))}{V}$ can be ignored. Therefore, according to the above analysis, under the premise of ensuring the data transmission constraints, each end device can get the optimal solution by making $T_i^d(t) = T_i^e(t)$ as much as possible.

5) Performance Analysis: This subsection presents the performance analysis of our algorithm.

Theorem 3. *By applying our methods, the time-average system delay satisfies:*

$$\lim_{T\to+\infty}\frac{1}{T}\sum_{T-1}^{0}\mathbb{E}\left[Y^*(t)\right] < Y^{opt} + \frac{B}{V} \tag{21}$$

where $Y^*(t)$ denotes the optimal average TCT in the t-th time slot under the proposed methods and Y^{opt} is the optimal time-average system average TCT.

Proof: See Appendix D. ■

IV. Experiment

A. Experiment settings

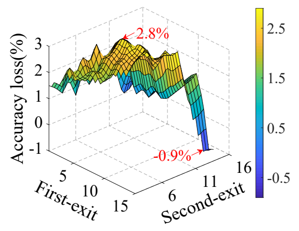
We build a prototype system to verify LEIME's performance. As shown in Fig. 5, our system consists of three parts: 1) the heterogeneous devices consist of 4 Raspberry Pis (3B+), and 2 NVIDIA Jetson Nanos. 2) the edge server is a desktop with i7-3770 CPU. 3) the cloud is equipped with Nvidia Tesla V100 GPUs. End devices communicate with the edge through WiFi, and the edge server link to the cloud with the Internet. We employ COMCAST [18] tool to control bandwidth and latency. Docker [19] container is adopted in our experiment to implement resource allocation and isolation in the edge server. The tasks are image recognition on the CIFAR-10 dataset [20], and the inference of Multi-exit DNN runs with Pytorch [21] deep learning framework. Experiments are carried out under four novel types of DNNs, VGG-16, Inception v3, ResNet-34, and SqueezeNet-1.0 DNN models. We have implemented the following mechanisms as benchmarks.

(1) DDNN [22]: Exits are set at the layers with a smaller amount of intermediate data and a higher exit probability.

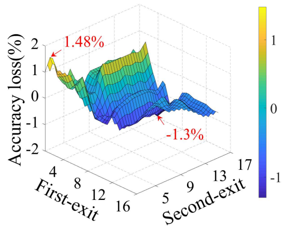
(2) Neurosurgeon [23]: DNN has no early exit, while the position of model partition is the same as that of LEIME.

(3) Edgent [24]: Exits are intuitively set at the position where intermediate data size is the smallest.

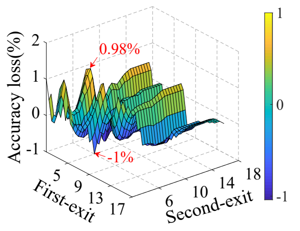
The above three benchmarks do not consider task offloading. Therefore, the offloading ratios of benchmarks are fixed to 0.



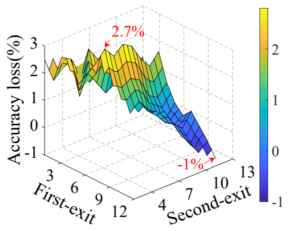
(a) The accuracy loss of ME-Inception v3.



(b) The accuracy loss of ME-ResNet-34.



(c) The accuracy loss of ME-SqueezeNet-1.0.



(d) The accuracy loss of ME-VGG-16.

Fig. 6. The accuracy loss of ME-DNNs.

B. Experiment Results

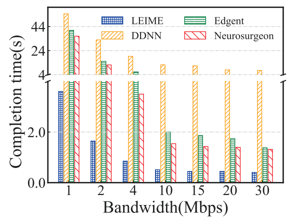
Test Case 1 (ME-DNN accuracy loss): In Fig. 6, we show the accuracy loss of ME-DNNs for Inception v3, ResNet-34, SqueezeNet-1.0, and VGG-16. We verify the accuracy loss of all possible combinations of the First-exit, Second-exit, and Third-exit (fixed as the last exit). The results indicate that compared with the corresponding original DNNs, the average accuracy loss of Multi-exit Inception v3 (ME-Inception v3), Multi-exit ResNet-34 (ME-ResNet-34), Multi-exit queezeNet-1.0 (ME-SqueezeNet-1.0), and Multi-exit VGG-16 (ME-VGG-16) are 1.62%, 0.55%, 0.44%, and 1.14%. We can see from the Fig. 6 that the accuracy loss is less than 0 in some case. It means that the accuracy of ME-DNNs is improved compared to the original DNN. For ResNet-34 and SqueezeNet-1.0, we can observe that most exit combinations obtain an accuracy increase of close to 1%. This phenomenon confirms a prevalent weakness of DNNs, "overthinking", proposed by Kaya Y. et al. [25], that is, DNNs will perform lots of redundant and wasteful calculations on some simple tasks, which eventually leads to a reduction in inference accuracy. Different DNNs have different model structures, leading to different "overthinking" situations. For example, for ME-Inception v3 and VGG-16, we can achieve accuracy improvement when both First and Second-exits are selected at deeper layer of the network, whereas, ME-ResNet-34 and SqueezeNet-1.0 turn to prefer lower layer exits. In conclusion, the use of ME-DNNs has little impact on accuracy in practical applications, sometimes can improve the accuracy of DNNs in turn.

Test Case 2 (Overall system performance): Fig. 7 shows the effect of networks on the average TCT. We use the trained Multi-exit Inception v3 model on Raspberry Pi to measure the average TCT when varying the propagation latency and bandwidth. Compared with Neurosurgeon, Edgent, and DDNN, LEIME upgrades $4.4\times$, $6.5\times$, and $18.7\times$ under different bandwidth and $4.2\times$, $5.7\times$, and $14.5\times$ under different propagation latency on average. Fig. 7 highlights that LEIME outstands the baseline methods, especially when the network condition is poor, such as, propagation latency is high ($> 100ms$) and bandwidth is small ($< 10Mbps$). We conclude that the best performance of LEIME comes from that we take the network conditions into consideration.

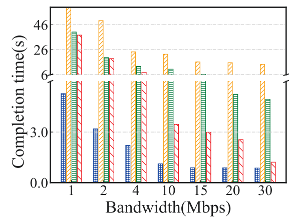
In Fig. 8, we evaluate the performance of LEIME under different DNN models (SqueezeNet-1.0, VGG-16, Inception v3, ResNet-34). Compared to the benchmarks, LEIME achieves $1.6\times$ to $13.2\times$ speedup on the Raspberry Pi, and $1.1\times$ to $10.3\times$ speedup on Nano. We can find that the performance change situation of Neurosurgeon is similar to that of LEIME, because the exit settings of these two methods are the same. However, the performance of Neurosurgeon is worse than that of LEIME because Neurosurgeon does not consider the early exit mechanism. For Edgent and DDNN, their time fluctuates greatly, because they intuitively set the exits and do not fully consider the characteristics of the DNNs. Therefore, their performance varies widely under different DNNs.

Test Case 3 (The stability of system): In Fig. 9, we evaluate the performance of LEIME under dynamic task arrival rates to explore the stability of LEIME. We observe that LEIME shows the smallest average TCT and best stability compared with the three benchmarks on both Raspberry Pi and Jetson Nano. We can find that the average TCT of Edgent on Raspberry Pi fluctuates greatly with the arrival rate, but not on Jetson Nano. This is because Jetson Nano has more powerful computational capability than Raspberry Pi, thus the computation of end devices is no longer a performance bottleneck. The curves for DDNN exceeds the y-axis range in Fig. 9(upper), but not in Fig. 9(lower). Because the computational capability of Raspberry Pi is limited, tasks cause a serious backlog in the end device queue. For Neurosurgeon, its curve fluctuates the most because it does not consider early exit and task offloading, resulting in a backlog of queues and an inability to balance the load between the end device and edge server.

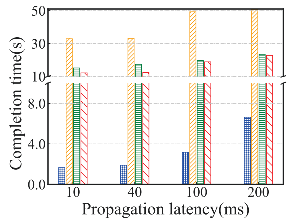
Test Case 4 (Algorithm evaluation): To verify our exit setting algorithm, in Fig. 10(a), we fixed the LEIME's task offloading algorithm as the task offloading mechanism and take three exit setting mechanisms based on minimization computation (min_comp), minimization transmission (min_tran), and average division (mean) as baselines. Overall, our exit setting algorithm is better than the other three strategies. Specifically, we can observe that, compared with the baselines, the speedup of LEIME for the large model (Inception v3 and ResNet-34) is better than that of the small model (SqueezeNet-1.0 and VGG-16). This is because, for small models, their intermediate data size and total FLOPs are relatively small. Both the transmission and computation time cost will not be a performance bottleneck. Moreover, we discover that the strategy of minimizing transmission overhead is generally the worst since it neglects the distribution of computation in the neural network layers, which is a sub-optimal strategy for



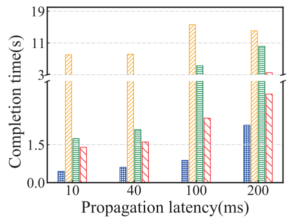
(a) Propagation latency=10ms.



(b) Propagation latency=100ms.

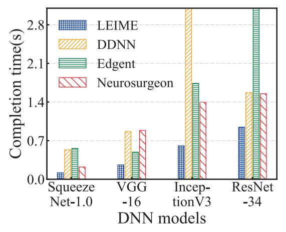


(c) Bandwidth=2Mbps.

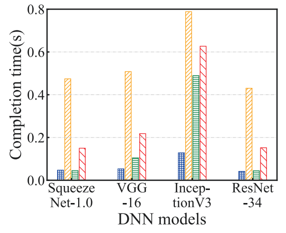


(d) Bandwidth=20Mbps.

Fig. 7. The effect of networks on average TCT.

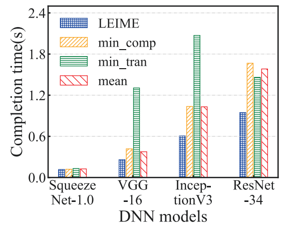


(a) Raspberry Pi 3B+.

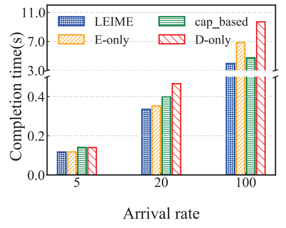


(b) Jetson Nano.

Fig. 8. The effect of different DNN models on average TCT.



(a) The effect of exit selection.



(b) The effect of task offloading.

Fig. 10. The effect of algorithms on average TCT.

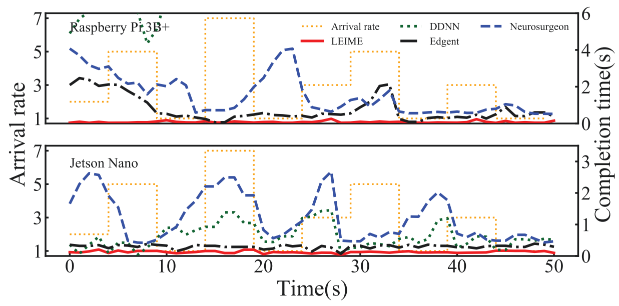
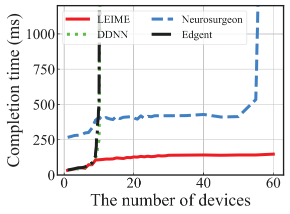
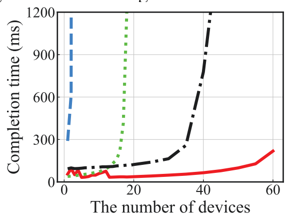


Fig. 9. The effect of task arrival rate on average TCT.



(a) Inception v3.



(b) ResNet-34.

Fig. 11. The effect of the number of devices on average TCT.

minimizing the cost.

In Fig. 10(b), we explore the offloading algorithm performance using Jetson Nano as end device. We compared LEIME's offloading algorithm with classical offloading algorithms, including device-only (D-only, offloading ratio is 0), edge-only (E-only, offloading ratio is 1), and capability-based (cap_based) offloading (based on the capability of device and edge). We can observe that when the task arrival rate is low (5 and 20), the performance of LEIME improves $1.1\times$ and $1.2\times$ compare with the baselines on average. However, when the task arrival rate is high (100), the performance of LEIME is significantly improved to $1.8\times$ on average. We can conclude that LEIME can efficiently adapt to the variety of task arrival, attributed to a dynamical adjustment of task offloading ratio in our online offloading algorithm.

Test Case 5 (The effect of the number of connected devices): In Fig. 11, we study the effect of the number of connected devices on average TCT in simulations based on the genuine parameter of Inception v3 and ResNet-34. We fixed the computational capabilities of edge servers and end devices, and assumed that all end devices are homogeneous. The results indicate that different from the benchmarks, as the number of connected devices increases, the average TCT of LEIME almost grows linearly. The proposed algorithms can achieve the lowest average TCT and support the most number of devices compared with three benchmark schemes. From our analysis, the reason is that LEIME's exit settings not only consider the characteristics of the DNNs, but also the load of the edge server. the optimal exit combinations will change to relieve the edge server load as the number of end devices increases. In this way, LEIME can support more device connections and reduce the average completion time of tasks.

V. Related Work

The literature of low latency DNN inference can be divided into the following 2 directions.

DNN partition: Yiping Kang et al. [23] proposed Neurosurgeon, a light-weight scheduler to partition DNN computation among mobile devices and data centers at the granularity of layers. Furthermore, DADS [4] proposed a dynamic DNN surgery scheme to split a DNN into two partitions to execute the DNN at the edge and cloud. Thaha Mohammed et al. [5], proposed an adaptive DNN partitioning scheme (DINA) based on matching theory. Although the above work can speed up the inference of DNN, it still cannot meet the ultra latency-sensitive application requirements due to a large amount of calculation of DNN models.

Multi-exit DNNs: As a result, multi-exit DNNs were proposed in recent researches [8][22][24] to further reduce the task inference time. BranchyNet [8] initially presented an architecture, to allow prediction to exit the network early via branches when certain confidence is achieved. Based on BranchyNet, S. Teerapittayanon et al. [22] attempted to apply this model to distributed deep learning applications. Furthermore, Edgent [24] proposed to leverage edge computing for DNN collaborative inference through device-edge synergy. The above works intuitively perform exit settings and does not take into account some dynamic factors, failing to deliver low latency and consistent performance. In contrast, this paper comprehensively analyzes the factors affecting performance and adaptively sets to optimal exits of ME-DNNs. More importantly, this work propose an online task offloading algorithm to fine-tune the task dispatching at runtime to resolve dynamic factors.

VI. Conclusion

In this paper, we proposed LEIME: at the model-level, we propose an exit setting algorithm to automatically build an optimal ME-DNN; at the computation-level, we present an online distributed wild-edge-aware offloading mechanism to fine-tune the task dispatching at runtime to maintain high performance when the tailored ME-DNN transient mismatch the dynamic environment. Real-world test evaluations demonstrate that LEIME significantly improves the application's performance, achieving $1.1 - 18.7\times$ speedup under different situations compared to baselines and remaining stable in the wild environment.

Appendix

A. Proof of the Theorem 2

We consider a DNN model with m layers and the last $exit_m$ is fixed as Third-exit. The $exit_{m-1}$ must not be First-exit, due to the First-exit should be shallower than the Second-exit. Therefore, we need to select the First-exit $exit_k$ from $exit_1$ to $exit_{m-2}$ in the first iteration, which requires $m-3$ comparison operations. When $exit_k$ is found, the exit with the least cost need to be found from $exit_{k+1}$ to $exit_{m-1}$ as a candidate Second-exit, which requires $m-1-k-1$ comparison operations. Therefore, we give the following recursive expression:

$$T(m) = m - 3 + \frac{1}{m-2}\sum_{k=1}^{m-2}[G(k) + m - 1 - k - 1] \quad (22)$$

where $G(k)$ denote the iteration from the second round.

When the $exit_k$ is founded, the next iteration need find the $exit_t$ from $exit_1$ to $exit_k - 1$, which requires $k-2$ comparison operations. Besides, it need to find the minimum Second-exit from $exit_{t+1}$ to $exit_{m-1}$, which requires $m-t-2$ comparison operations. We give the recursive expression of $G(k)$.

$$G(k) = k - 2 + \frac{1}{k-1}\sum_{t=1}^{k-1}[G(t) + m - t - 2] \quad (23)$$

Based on the recursive expression (23), we can derive the following expression.

$$G(k) \leq k - 1 + (n-4)[ln(k-1) + C] \quad (24)$$

where $k \geq 2$ and C is a constant.

By plugging (24) into (22) we can obtain:

$$\begin{aligned}
T(m) &\leq 2m - 6 + \frac{m-4}{m-2}\sum_{k=2}^{m-2}[ln(k-1) + C] \\
T(m) &\leq 2m - 6 + \frac{C(m-4)}{m-2} + \frac{m-4}{m-2}\sum_{k=2}^{m-2} ln(k-1) \\
T(m) &\leq 2m - 6 + \frac{C(m-4)}{m-2} + \frac{m-4}{m-2} ln((m-3)!) \\
T(m) &\leq 2m - 6 + \frac{C(m-4)}{m-2} + \frac{m-4}{m-2} mln(m)
\end{aligned} \quad (25)$$

Therefore, the time complexity is $O(mln(m))$

B. The resource allocation strategy of edge server

The computing resources available to the i-th end device can be expressed as $F_i^d + p_i F^e$. And for the i-th device, the amount of FLOPs that need to be performed on end devices and edge of each time slot is $k_i(\mu_1 + (1-\sigma_1)\mu_2)$. Thus, the mean processing time of tasks in each time slot can be expressed as $(k_i(\mu_1 + (1-\sigma_1)\mu_2))/(F_i^d + p_i F^e)$. Therefore, the average task inference time in the whole system is:

$$f(P) = \frac{1}{\sum_{i\in\mathcal{N}} k_i}\left(\sum_{i\in\mathcal{N}} \frac{k_i(\mu_1 + (1-\sigma_1)\mu_2)}{F_i^d + p_i F^e}\right) \quad (26)$$

where $P \triangleq \{p_i | i \in \mathcal{N}\}$.

We get the resource allocation scheme by minimizing $f(P)$. The minimizing $f(P)$ can be proved as a convex optimization problem. With the Karush-Kuhn-Tucker (KKT) conditions [17], the optimal solution of $P \triangleq \{p_i | i \in \mathcal{N}\}$ is calculated as:

$$p_i = \sqrt{\frac{k_i}{\left(\frac{F^e \sum_{i=1}^n \sqrt{k_i}}{\sum_{i=1}^n F_i^d + F^e}\right)^2}} - \frac{F_i^d}{F^e}, \forall i \in \mathcal{N} \quad (27)$$

C. Proof of the Lemma 1

Proof: Due to $max\{Q_i(t) - b_i(t), 0\}^2 \leq (Q_i(t) - b_i(t))^2$:

$$\begin{aligned}
\Delta(Q(t)) &= \frac{1}{2}\mathbb{E}\left[Q_i^2(t+1) - Q_i^2(t)|Q(t)\right] \leq \frac{1}{2}(Q_i(t) - \\
&b_i(t))^2 + \frac{1}{2}A_i(t)^2 - Q_i^2(t) + max\{Q_i(t) - b_i(t), 0\}A_i(t) \\
&= \frac{A_i^2(t) + b_i^2(t)}{2} - \widetilde{b}_i(t)A_i(t) + Q_i(t)(A_i(t) - b_i(t))
\end{aligned} \quad (28)$$

where $\widetilde{b}_i(t) = min[Q_i(t), b_i(t)]$,therefore $\Delta(Q(t)) \leq B_1 + Q_i(t)(A_i(t) - b_i(t))$, where $B_1 = max\{\frac{A_i^2(t)+b_i^2(t)}{2} - \widetilde{b}_i(t)A_i(t)\}$. Similarly, we can also get $\Delta(H(t)) \leq B_2 + Q_i(t)(D_i(t) - C_i(t))$, where $B_2 = max\{\frac{D_i^2(t)+c_i^2(t)}{2} - \widetilde{c}_i(t)D_i(t)\}$. Finally, let $B = B_1 + B_2$, we can get $\Delta(\mathbf{\Theta}(t)) \leq B + Q_i(t)(A_i(t) - b_i(t)) + H_i(t)(D_i(t) - c_i(t))$. ■

D. Proof of the Theorem 3

Proof: We first introduce the Lemma 2, which is obtained from Theorem 4.5 of [9].

Lemma 2. *For any $\delta > 0$, there exists a stationary and randomized policy $X^*(t)$ for $P4$, which decides $Y^*(t), A^*(t), D^*(t), b^*(t), c^*(t)$ are independent of the current queue backlogs $\mathbf{\Theta}(t)$, such that the following inequalities are satisfied:*

$$\mathbb{E}\{Y^*(t)|\mathbf{\Theta}(t)\} = \mathbb{E}\{Y^*(t)\} \leq Y^{opt} + \delta \tag{29}$$

$$\mathbb{E}\{A^*(t) - b^*(t)|\mathbf{\Theta}(t)\} = \mathbb{E}\{A^*(t) - b^*(t)\} \leq \delta \tag{30}$$

$$\mathbb{E}\{D^*(t) - c^*(t)|\mathbf{\Theta}(t)\} = \mathbb{E}\{D^*(t) - c^*(t)\} \leq \delta. \tag{31}$$

By plugging Lemma 2 into the $drift - plus - penalty$ inequality (17) and taking $\delta \to 0$ we obtain:

$$\Delta(\mathbf{\Theta}(t)) + V\mathbb{E}\left[Y^*(t)\right] \leq B + VY^{opt} \tag{32}$$

We sum the inequality over $t \in \{0, 1, ..., T-1\}$ and divide the result by T to get:

$$\begin{aligned}\frac{1}{T}\mathbb{E}[L(\mathbf{\Theta}(t)) - L(\mathbf{\Theta}(0))] + \frac{V}{T}\sum_{T-1}^{0}\mathbb{E}\left[Y^*(t)\right] \\ < VY^{opt} + B\end{aligned} \tag{33}$$

Obviously, formula (27) is obtained by solving the convex optimization problem, thus $P \triangleq \{p_i | i \in \mathcal{N}\}$ is an optimal value. Therefore, we can divide both side of inequality (33) by V and take a lim sup as $T \to +\infty$ to prove Theorem 3. ∎

Acknowledgment

This work is supported by National Key R&D Program of China 2018AAA0100500, National Natural Science Foundation of China under Grants, No. 61872079, 61902065, 61632008, 61702096, 61702097, 61972085, 61906040, the Natural Science Foundation of Jiangsu Province under grant BK20170689, BK20190345, BK20190335, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No.BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No.93K-9, the University Synergy Innovation Program of Anhui Province No. GXXT-2020-012, and partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization and Collaborative Innovation Center of Wireless Communications Technology. We also thank the Big Data Computing Center of Southeast University for providing the experiment environment and computing facility.

References

[1] L. Liu, B. Wu, and W. Shi, "A comparison of communication mechanisms in vehicular edge computing," in *HotEdge*, 2020.

[2] G. Guo and N. Zhang, "A survey on deep learning based face recognition," *Comput. Vis. Image Underst.*, vol. 189, 2019.

[3] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, pp. 1738–1762, 2019.

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

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

[6] X. Tang, X. Chen, L. Zeng, S. Yu, and L. Chen, "Joint multiuser dnn partitioning and computational resource allocation for collaborative edge intelligence," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[7] W. He, S. Guo, S. Guo, X.-S. Qiu, and F. Qi, "Joint dnn partition deployment and resource allocation for delay-sensitive deep learning inference in iot," *IEEE Internet of Things Journal*, vol. 7, pp. 9241–9254, 2020.

[8] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, 2016.

[9] J. Michael, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211.

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

[11] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, 2009.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 2818–2826.

[14] "Raspberry pi," https://www.raspberrypi.org.

[15] "Jetson nano," https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[16] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*. IEEE, 2018, pp. 207–215.

[17] Boyd, Vandenberghe, and Faybusovich, "Convex optimization," *IEEE Transactions on Automatic Control*, vol. 51, no. 11, pp. 1859–1859, 2006.

[18] "Comcast," https://github.com/sendgrid/comcast.

[19] "Docker," https://www.docker.com/.

[20] "Cifar-10," http://www.cs.toronto.edu/ kriz/cifar.html.

[21] "Pytorch," https://pytorch.org/.

[22] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 328–339, 2017.

[23] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. N. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.

[24] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.

[25] Y. Kaya, S. Hong, and T. Dumitras, "Shallow-deep networks: Understanding and mitigating network overthinking," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3301–3310.