DE^{2}R: Unifying DVFS and Early-Exit for Embedded AI Inference via Reinforcement Learning

Yuting He, Jingjin Li, Chengtai Li, Qingyu Yang, Zheng Wang, Heshan Du, Jianfeng Ren, Heng Yu

Abstract—**Executing neural networks on resource-constrained embedded devices faces challenges. Efforts have been made at the application and system levels to reduce the execution cost. Among them, the early-exit networks reduce computational cost through intermediate exits, while Dynamic Voltage and Frequency Scaling (DVFS) offers system energy reduction. Existing works strive to unify early-exit and DVFS for combined benefits on both timing and energy flexibility, yet limitations exist: 1) varying time constraints that make different exit points become more, or less, important in terms of inference accuracy, are not taken care of, and 2) the optimal decisions of unifying DVFS and early-exit as a multi-objective optimization problem are not achieved due to the large configuration space. To address these challenges, we propose DE^{2}R, a reinforcement learning-based framework that jointly optimizes early-exit points and DVFS settings for continuous inference. In particular, DE^{2}R includes a cross-training mechanism that fine-tunes the early-exit network to accommodate dynamic time constraints and system conditions. Experimental results demonstrate that DE^{2}R achieves up to 22.03% energy reduction and 3.23% accuracy gain compared to contemporary techniques.**

Index Terms—**Reinforcement Learning, Early-Exit Neural Networks, DVFS, Embedded Computing.**

I. INTRODUCTION

Continuous up-scaling of deep neural network models leads to increased inference latency and deployment challenges on resource-constrained embedded devices. Approaches have been designed for neural network light-weighting, both spatially [6], [17] and temporally [5], [21]. Among them, the early-exit neural network [21], as a type of temporal approach, is designed to allow the model to terminate computation early when sufficient inference confidence is reached. It introduces multiple exit points at different layers of a neural network, and is able to dynamically adjust inference time, accuracy, and computational resource utilization, making it a promising approach for embedded AI realization.

System energy reduction, as another key optimization metric of embedded execution, is still a side effect along with inference time/accuracy reduction in early-exit networks. On the opposite, Dynamic Voltage and Frequency Scaling (DVFS) [12], [25] has long been recognized as an energy-oriented technique that leads to time adjustment. Considering either individually could bring compromise in overall optimization of time, energy, and accuracy for neural network inference. Thus, there exists an opportunity to unify DVFS and early-exit, which both provide dynamism in neural network inference, to achieve orchestrated optimization for the above metrics. For example, under tight time constraints, a model may choose a higher V-F setting to speed up the inference, allowing complex inputs to exit from rear layers, thereby improving accuracy. In contrast, with relaxed time constraints, reducing V-F could save energy at the same level of inference accuracy. However, as implied in the example, the unification faces challenges: 1) The optimization still involves conflicting objectives – high V-F achieves improved inference accuracy at the price of increased energy, for example. 2) Co-considering DVFS and early-exit presents a vast configuration space. 3) The system environment is highly dynamic, with varying inputs, system status, and inference deadlines that require adaptable decision-making strategies.

A few endeavors have made efforts so far to tackle these challenges. For example, [13] uses bag-of-feature pooling to evaluate input difficulty and determine the exit point, and then adjust the V-F based on the individual input's time constraint and exit point. However, there still remain improvement space for the state of the art: 1) For different time constraints, the importance of each exit varies. For example, under tight deadlines, all inputs may need to exit at the earliest exit point, making the accuracy of existing from there more important. This implies a requirement to actively adapt the network's parameters to varying time constraints, in other words, network re-training for early exiting purpose. 2) The decision-making heuristics proposed in existing methods, such as [13], chooses the exit point and V-F setting individually or interleavingly, potentially leading to suboptimal decisions.

To address these challenges, in this paper, we design a reinforcement learning (RL)-based method to optimize early-exit neural network accuracy and energy consumption within specific time constraints in continuous inference scenarios, namely DE^{2}R. RL is a data-driven method that can handle large search spaces and effectively manage multi-objective optimization. Specifically, to tackle the challenge of accommodating to various time constraints, we introduce a cross-training mechanism to allow the network to be retrained accordingly. In our method, to resolve the suboptimal decisions caused by the interleaving decision-making process, an RL agent is

This work was supported by Zhejiang Natural Science Foundation under grant LY24F020006, and National Nature Science Foundation of China under grants 62220106011 and 62372442.

Yuting He is with the School of Computer Science, University of Nottingham Ningbo China, Ningbo, China. She is also with Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China. Email: yuting.he@nottingham.edu.cn

Jingjin Li, Chengtai Li, Qingyu Yang, Heshan Du, Jianfeng Ren, and Heng Yu are with the School of Computer Science, University of Nottingham Ningbo China, Ningbo, China. Emails: {jingjin.li, chengtai.li, qingyu.yang, heshan.du, jianfeng.ren, heng.yu}@nottingham.edu.cn. Corresponding author: Heng Yu.

Zheng Wang is with Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China. Email: zheng.wang@siat.ac.cn

proposed to simultaneously determine both the exit point and V-F settings. The agent is designed to accommodate dynamic environments, such as varying inputs, total inference time, and the evolving parameters of the early-exit neural networks. Our contributions are summarized as follows:

- We propose DE^{2}R, a novel RL-based method to judiciously select both DVFS settings and early-exit points in continuous inference, to jointly optimize inference accuracy and energy consumption under time constraints.
- We introduce a cross-training mechanism to further improve the inference accuracy for early-exit networks, accommodating varying conditions such as time constraints.

We compare with contemporary techniques and perform ablation studies under various neural network models, datasets, and time constraints, demonstrating that DE^{2}R achieves up to 22.03% energy reduction and 3.23% accuracy gain compared to contemporary techniques.

The rest of the paper is organized as follows. Section II reviews the existing works on early-exit neural network architectures, exiting strategies, and integration with DVFS. Section III provides preliminaries on early-exit neural networks. Section IV describes the proposed DE^{2}R framework. Section V presents the cross-training mechanism. Section VI covers the experimental setup, result, and analysis. Section VII summarizes the key findings of this paper.

II. Related Work

There have been works designing early-exit network architectures. For example, [16] proposes hierarchical early exits for deep convolutional neural networks, and [23] proposes to reuse the predecessor's prediction to prevent the waste of computation, by inserting direct connection between layers. Early exit strategies are also combined with other techniques for further network acceleration. [9] constructs an efficient early-exit network with hardware-friendly dynamic branch pruning, and [20] designs an edge-cloud hierarchical training mechanism to accelerate the training and reduce data transmission.

Various strategies have been proposed to determine each input's exit. [21] proposes confidence-based exit criteria, where the highest probability in the output distribution is considered as confidence. [26] raises patience-based exit criteria, where input can exit early if a certain number of successive exits give the same prediction. RL-based strategies for early-exiting are also observed. [8] adopts Q-network to select the optimal exit for each input. [10] uses a contextual multi-armed bandit algorithm to dynamically select the optimal exit point during inference. [7] combines quantization and early-exit, using reinforcement learning to optimize layer-wise bit-width and exit point placement. At the system level, [24] studies thermal management for adaptive applications, and [3] proposes an RL-based DVFS approach to optimize application quality on energy-harvesting devices. Recent works, such as [13], combine early-exit and DVFS for embedded AI, but struggle with multi-objective optimization of accuracy, energy, and time.

III. Preliminaries

Early-exit neural networks are a type of dynamic neural network that allows for intermediate predictions at different exit points within the model, rather than requiring the input to pass through all layers. Given an early-exit network M_ψ parameterized by ψ with M exits, the auxiliary branches are denoted as $M_\psi^1, M_\psi^2, \ldots, M_\psi^m$. For a given input x, the energy consumption $E(m)$ of inferring x through the m-th exit point is calculated by summing the energy consumed by each layer along the inference path of M_ψ^m, formulated as,

$$E(m) = \sum_{i=1}^{L_m} e_i, \tag{1}$$

where e_i is the energy consumption of the i-th layer, L_m is the total number of layers on inference path of M_ψ^m. In this work, we measure the energy consumption $E(m)$ for each inference path on an embedded CPU. For classification tasks, the overall accuracy Acc of inferring N^{total} inputs in the early-exit network is defined as,

$$Acc = \frac{1}{N^{total}} \sum_{i=1}^{m} N_i A_i, \tag{2}$$

where N^{total} is the total number of inputs, N_i is the number of inputs that exit at the i-th exit point M_ψ^i, and A_i is the average inference accuracy of exited inputs at the i-th exit point.

IV. RL-based DVFS and Early-Exit (DE^{2}R)

Fig. 1 shows an overview of our proposed method and its targeted architecture. Overall, DE^{2}R is built on an RL framework consisting of the environment and agent. The environment is composed of an early-exit network, a system monitor, and a DVFS governor. The early-exit network enables intermediate predictions at multiple exit points, while the DVFS governor adjusts the processor's V-F dynamically based on system conditions. The system monitor offers real-time metrics on energy consumption, task completion, and inference performance to aid dynamic decision-making. The agent consists of a *difficulty encoder* and an actor-critic structure for real-time decision-making. The *difficulty encoder*, comprising a convolutional layer, average pooling, and a sigmoid function, predicts input complexity as a scalar $D \in [0, 1]$. The actor determines early-exit points and V-F adjustments, while the critic evaluates actions by estimating state values to refine the actor's decisions.

The underlying multi-objective optimization problem is formulated as a Markov Decision Process (MDP), denoted as $M = (S, A, P, R, \gamma)$, with the following components: (1) **State** S represents a set of all possible states of the system. The state s_t at time t is denoted as $s_t = \{D_t, V_t, F_t, Acc_t^c, T_t^r, N_t^r, E_t^c\}$, and the definition and range of each state attribute are explained in Table I. (2) **Action** A represents a set of all possible actions that the agent can take at each state. We design $a_t = (a_t^{exit}, a_t^{ps})$, where a_t^{exit} determines the selected exit point in the early-exit network, while a_t^{ps} adjusts the processor's V-F settings. (3) **State transition** $P(s'|s, a)$ represents the probability of

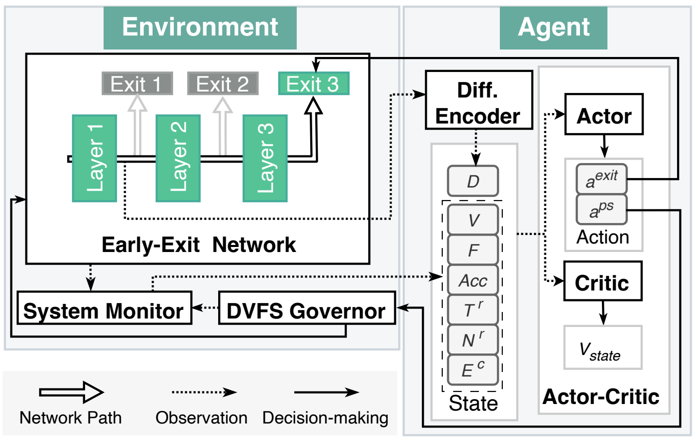


Fig. 1. Overview of DE^{2}R. The environment includes the early-exit neural network, DVFS governor, and system monitor. The agent consists of an actor-critic framework for decision-making and state evaluation.

transitioning from state s to state s' after taking action a. (4) **Reward** $R(s, a)$ represents the immediate reward vector received after taking action a at state s. In DE^{2}R, we apply the concave-augmented reward [14] for computing the scalarized immediate reward $R(s, a)^s$. To evaluate the action selection, we design a reward function using linear scalarization that can reflect the instant inference accuracy Acc_t, energy consumption E_t, and time consumption of each input's inference. Denote the immediate reward vector as $R(s_t, a_t) = [Acc_t, -E_t]$, the immediate scalarized reward is computed as follows,

$$R(s_t, a_t)^s = \boldsymbol{w}^T R(s_t, a_t) + \max(0, -T^r)P + \alpha f(\pi(s_t, \boldsymbol{w})) \tag{3}$$

where $\boldsymbol{w}$ is a weight vector with $\|\boldsymbol{w}\|_1 = 1$ used to balance different objectives in the reward function, P is a large penalty, typically -10^3, which is applied when the time constraint is not satisfied, and f is a strongly concave term. Following [14], we set f to be the entropy of the action probability,

$$H(\pi(\cdot|s)) = -\sum_{a \in \mathcal{A}} \pi(a|s) \log \pi(a|s) \tag{4}$$

The concave-augmented reward function provides stability and smoothness to the learning process, ensuring more consistent policy optimization. Using entropy as the concave term encourages exploration by promoting a more uniform action distribution, preventing the early convergence to suboptimal policies. (5) **Discount factor** γ represents the discount rate of the future reward, ranging from 0 to 1. The agent makes decisions according to a policy π, which is a strategy that specifies the action to take in each state. The optimal policy, denoted as π^*, determines the best action to take from any given state to achieve the highest expected cumulative reward. In this formulation, RL allows the system to learn an optimal policy π^* that accommodates changing conditions, maximizing the cumulative reward by intelligently selecting exit points and V-F settings.

During the continuous inference process, the agent observes the system state s_t at each time step. Based on this state, the agent selects an action $a_t = (a_t^{exit}, a_t^{ps})$. The environment, which includes the early-exit network and system conditions, responds to the agent's actions, updating the state and providing a reward signal based on the current energy consumption, accuracy, and time. The agent continually learns from its actions, refining its policy over time to maximize accuracy, minimize energy consumption, and meet inference deadlines.

TABLE I
STATE ATTRIBUTES

Attr.	Range	Description
D	[0, 1]	Difficulty of Processing
V	(0, 1]	Normalized discrete core voltage
F	(0, 1]	Normalized discrete core frequency
Acc^c	[0, 1]	Accumulated accuracy
T^r	[0, 1]	Ratio of remaining time over total time of the continuous batch of NN tasks
N^r	[0, 1]	Ratio of remaining task over total no. of the continuous batch of NN tasks
E^c	$(0, +\infty)$	Accumulated consumed energy

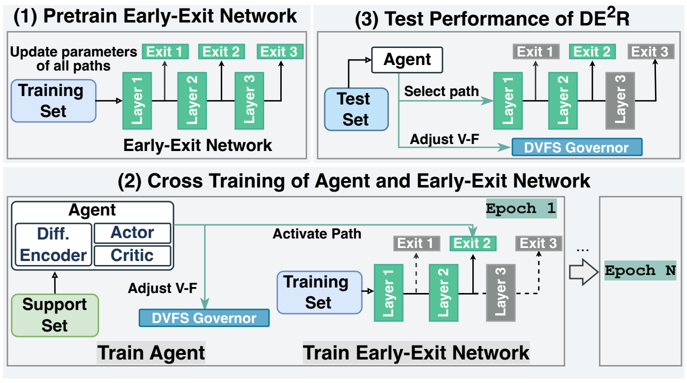


Fig. 2. The Cross-training mechanism. The process involves pretraining the early-exit network, cross-training the agent and the early-exit network, and testing the overall performance of DE^{2}R.

V. CROSS-TRAINING OF DE^{2}R

In Section IV, the early-exit network is assumed to have fixed parameters. However, under continuous inference scenarios, the importance of each exit may vary. For example, in situations where all inputs are forced to exit from the earliest exit point due to stringent time constraints, fine-tuning the layers of the shortest inference path can result in an accuracy improvement without increasing energy consumption. Based on this intuition, we propose the cross-training mechanism, which allows the agent to be trained with awareness of various situations that impose various customized parameters for an exit point, while the early-exit network is also fine-tuned accordingly. The overview of the cross-training mechanism is shown in Fig. 2. Given a pre-trained early-exit network, an alternating training process is designed within each epoch, where the agent is trained first, focusing on optimizing the policy and value networks, followed by fine-tuning the early-exit neural network. The policy is then tested with the trained early-exit network and the agent.

Trajectory collection. During training, since the policy learning and early-exit network fine-tuning rely on interactions between the agent and the environment, we first collect the trajectory data that represents various states, actions, and outcomes. **Algorithm 1** summarizes the process of trajectory collections. Given an early-exit network M_ψ with m branches, the total time constraint T^c of the continuous batch, and the number of inference tasks N, we firstly collect trajectory on a pre-trained early-exit network by setting the initial state s_0 to $Acc_t^c = 0, T_t^r = 1, N_t^r = N, E_t^c = 0$, and D_t, V_t and F_t randomly, where $t = 0$. At each time step t, forward input x_t through M_ψ before M_ψ^1, where two inference paths diverge and the network should make the decision regarding whether to terminate computation. The input difficulty D_t is computed through the difficulty encoder, which helps to decide the difficulty of the input for further decision. Then, the agent observes the updated state s_t, and the actor chooses the action $a_t = \pi_\theta(s_t)$ based on the policy network π_θ while the agent critic computes the state value $V_\phi(s_t)$ through the value network V_ϕ. When all inputs complete inference within T^c, the environment is set to idle until T^c. The total energy consumption up to time t is calculated as $E_t^c + E^{idle}$. The scalarized reward is computed according to Eqn. (3). Trajectory τ is appended by $(s_t, a_t, \pi_\theta(a|s), r_t, s_{t+1})$.

Algorithm 1 Trajectory Collection for DE^{2}R

```
Require: Pre-trained early-exit network $M_\psi$, policy network $\pi_\theta$,
    power state $P_1, P_2, ..., P_y$, total time constraint $T^c$, number of
    inputs $N$ for each trajectory
 1: Initialize state $s_0$, $N^r = N$
 2: while $N^r > 0$ do
 3:     Forward input $x_t$ through $M_\psi$ before $M_\psi^1$
 4:     Compute input difficulty $D_t$
 5:     Agent observes state $s_t$
 6:     Agent chooses action $(a_t^{exit}, a_t^{ps}) = \pi_\theta(s_t)$
 7:     Compute reward vector $r_t = R(s_t, a_t)$
 8:     Append $(s_t, a_t, \pi_\theta(a|s), r_t, s_{t+1})$ to trajectory $\tau$
 9:     Update $V_t, F_t, Acc_t, T_t^r, N_t^r, E_t^c$
10: end while
11: if $T^r > 0$ then
12:     Set the environment to idle until $T^c$
13:     Compute idle energy $E^{idle}$
14:     $E_t^c + = E^{idle}$
15: end if
16: return Trajectory $\tau$
```

Agent update. After collecting the trajectory, we adopt the PPO-AC algorithm [19] as it provides a stable and efficient framework for learning in environments under complex multi-objective optimization problems. Specifically, the collected trajectory τ is divided into minibatch b to ensure more stable and efficient updates. For each $(s_t, a_t, \pi_\theta(a|s), r_t, s_{t+1})$ in b, the actor and critic are updated respectively. We first calculate the importance sampling weight $isw_t(\theta)$, which measures the difference between the new policy and the old policy in terms of the probability of selecting the same action in the current state. The $isw_t(\theta)$ is computed using the following equation:

$$isw_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{5}$$

Next, we compute the return R_t, which represents the cumulative discounted reward starting from time step t. By considering future rewards, the agent makes decisions that consider long-term benefits. The return R_t at time step t is calculated as,

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{6}$$

where γ is the discount factor, which balances the impact of future rewards and immediate rewards. Then, the advantage A_t of the selected action is computed. It reflects the additional benefit of choosing action a_t in a given state s_t compared to the average action selected by the current policy.

$$A_t = R_t - V_\phi(s_t), \tag{7}$$

where $V_\phi(s_t)$ is the state value function and is represented by a fully connected layer. It represents the expected total return when following the current policy $\pi_\theta(a_t|s_t)$ at state s_t, which acts as a baseline that allows the advantage to be calculated. We use PPO-CLIP to limit the update magnitude of the policy, thereby ensuring stability in the training process. The policy loss $\mathcal{L}_t^{\text{CLIP}}(\theta)$ is defined as,

$$\mathcal{L}_t^{\text{CLIP}}(\theta) = \min(isw_t(\theta)A_t, \text{Clip}(isw_t(\theta), 1-\epsilon, 1+\epsilon)A_t) \tag{8}$$

where ϵ is the clipping parameter. By minimizing $\mathcal{L}_t^{\text{CLIP}}(\theta)$, the policy is stably updated to maximize the total rewards. To update the value network $\mathcal{L}_t^{\text{V}}(\phi)$, we define the loss $\mathcal{L}_t^{\text{V}}(\phi)$, which measures the difference between the predicted state value $V_\phi(s_t)$ and the actual return R_t, as:

$$\mathcal{L}_t^{\text{V}}(\phi) = (V_\phi(s_t) - R_t)^2 \tag{9}$$

By minimizing $\mathcal{L}_t^{\text{V}}(\phi)$, the value network better estimates the expected future rewards in each state. By iteratively updating the policy network by $\mathcal{L}_t^{\text{CLIP}}(\theta)$ and value network by $\mathcal{L}_t^{\text{V}}(\phi)$, the agent gradually learns the optimal policy that maximizes the expected cumulative reward with a stable learning process.

Early-exit network fine-tuning. In addition to training the agent, we fine-tune the early-exit network within the environment to ensure it accommodates better to dynamic conditions. The cross-training algorithm of DE^{2}R is described in **Algorithm 2**. For each state s_t in the trajectory τ, the agent selects updated actions (a_t^{exit}, a_t^{ps}) under the current policy π_θ, and the input x_t is forwarded through the selected model path of $M_\psi^{a_t^{exit}}$, generating the model's output for the current state. The task inference loss $\mathcal{L}_{task}$ is then computed at the exit $M_\psi^{a_t^{exit}}$ to measure the model's performance on this path. The loss $\mathcal{L}_{task}$ is backpropagated to update the parameters of the inference path of $M_\psi^{a_t^{exit}}$, thereby minimizing $\mathcal{L}_{task}$ and improving the performance of the early-exit network. This process is repeated for each state in the trajectory τ. By iteratively fine-tuning it and altering the agent's policy updates, the overall performance and adaptability of the early-exit network can be improved.

VI. EXPERIMENTS

A. Setup

Our experiments are conducted by collecting data and performing simulations on MobileNetV2 [18], StarNet050 [15], and

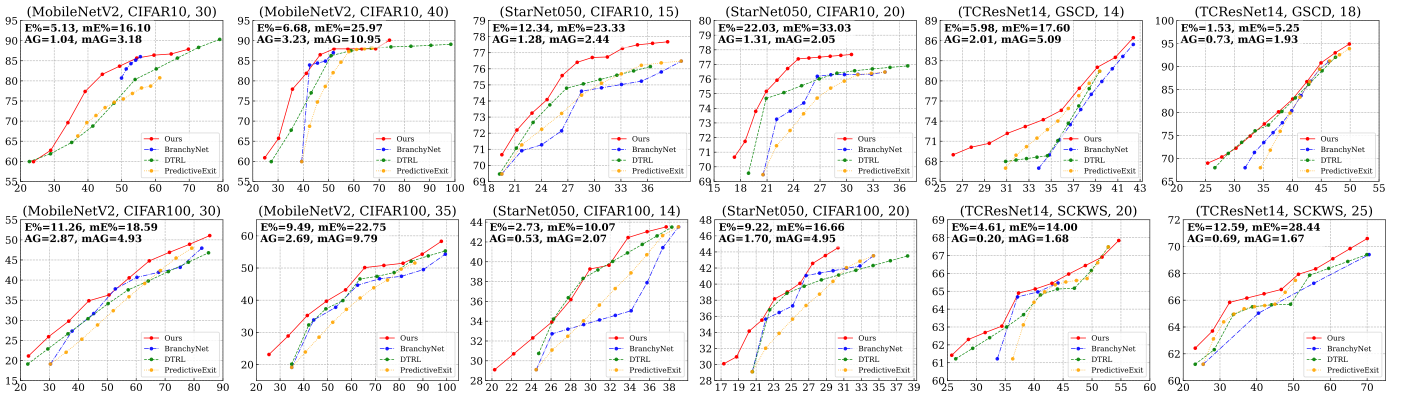


Fig. 3. Comparing DE^{2}R with state-of-the-art methods. Each data point represents one valid continuous inference, with the time constraint T^c not violated. Each subplot corresponds to a combination of (model, dataset, T^c(s)). The x-axes represent the energy consumption in joules (J), while the y-axes denote the accuracy of the model in percentage (%). Data approaching top-left represents that the inference results in high accuracy and low energy consumption.

Algorithm 2 Cross Training Process of DE^{2}R

```
Require: Pre-trained early-exit network $M_\psi$, power state $P_1, P_2, ..., P_y$, total time constraint $T^c$, discount factor $\gamma$, clip ratio $\sigma$, minibatch size $b$, policy network $\pi_\theta$, value network $V_\phi$,
 1: for each epoch do
 2:     Collect trajectory $\tau$ using Algorithm 1
 3:     // Policy Update
 4:     for $(s_t, a_t, \pi_\theta(a|s), r_t, s_{t+1})$ in minibatch do
 5:         Compute policy loss $\mathcal{L}_t^{\text{CLIP}}(\theta)$ // ref. Eqn. (8)
 6:         Compute value loss $\mathcal{L}_t^{\text{V}}(\phi)$ // ref. Eqn. (9)
 7:         Update $V_\phi$ and $\pi_\theta$ using Adam optimizer
 8:     end for
 9:     // Finetune early-exit network
10:     for each state $s_t$ in $\tau$ do
11:         Obtain updated actions $(a_t^{exit}, a_t^{ps}) = \pi_\theta(s_t)$
12:         Forward pass input $x_t$ through $M_\psi^{a_t^{exit}}$
13:         Compute task inference loss $\mathcal{L}_{task}$ at $M_\psi^{a_t^{exit}}$
14:         Backpropagate $\mathcal{L}_{task}$ through path to $M_\psi^{a_t^{exit}}$
15:         Update parameters of path to $M_\psi^{a_t^{exit}}$
16:     end for
17: end for
```

TCResNet14 [4] with 3, 4, and 2 exits respectively. For each model, we evaluate the performance across different datasets. Specifically, MobileNetV2 and StarNet050 are tested on CIFAR10 and CIFAR100 [11], while TCResNet14 is evaluated on the Google Speech Command Dataset (GSCD) [22] and ASR-SCKwsptSC (SCKWS) datasets [1]. Initially, the necessary data are collected on the ARM Cortex-A53 CPU, including the time and energy required for inference under different models, datasets, inference paths, and power states. We assume that data collection under the same settings forms a normal distribution. In the simulation environment, the time and energy required for inference are then obtained based on the probabilities defined by these normal distributions.

The early-exit network is first trained end-to-end on the training set. To enhance the generalization of the agent, we use a support set to train the agent instead of the training set, which prevents the potential overfitting of early-exit networks on the training set from affecting the agent's generalization ability. The hyper-parameters of DE^{2}R are shown in Table II.

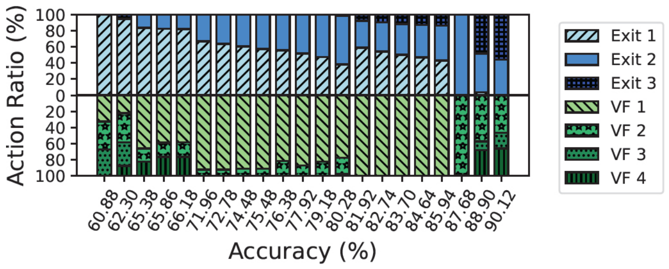
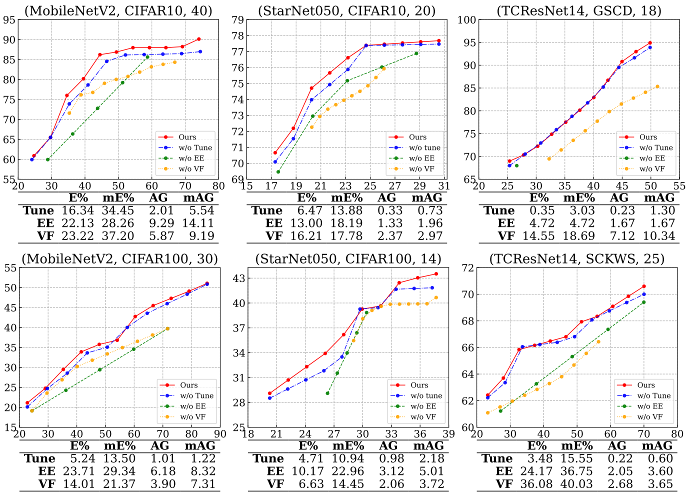


Fig. 4. Action ratio of (MobileNetV2, CIFAR10, 30) under different accuracy levels. Exit 1 corresponds to the earliest exit, and VF 1 represents the lowest V-F setting.

Compared methods. We compare DE^{2}R with three state-of-the-art methods. **BranchyNet** [21] is a classic early-exit neural network that enables early prediction by hand-tuning the exit threshold. To adapt it to our scenario, we select 50 equally spaced thresholds in the range of [0,1], with an interval of 0.02 between each threshold. As BranchyNet lacks DVFS adjustment, we experiment with different power states for all thresholds, presenting Pareto-optimal data points. **DTRL** [2] is a multi-objective variant of the Proximal Policy Optimization algorithm with a differentiable decision tree as policy representation. It is initially proposed to optimize the performance and energy consumption in runtime task scheduling. We adapt DTRL to our scenario by replacing the original states and actions with our own states and actions. **PredictiveExit** [13] introduces a heuristic for fine-grained early-exit prediction combined with DVFS. It sets a fixed time constraint for the inference time of a single input, predicts the optimal exit point through bag-of-feature pooling during inference, and adjusts the system's power states accordingly to optimize energy consumption. We compare our method with the above-mentioned methods in terms of accuracy and energy consumption.



(MobileNetV2, CIFAR10, 40)

	E%	mE%	AG	mAG
Tune	16.34	34.45	2.01	5.54
EE	22.13	28.26	9.29	14.11
VF	23.22	37.20	5.87	9.19

(StarNet050, CIFAR10, 20)

	E%	mE%	AG	mAG
Tune	6.47	13.88	0.33	0.73
EE	13.00	18.19	1.33	1.96
VF	16.21	17.78	2.37	2.97

(TCResNet14, GSCD, 18)

	E%	mE%	AG	mAG
Tune	0.35	3.03	0.23	1.30
EE	4.72	4.72	1.67	1.67
VF	14.55	18.69	7.12	10.34

(MobileNetV2, CIFAR100, 30)

	E%	mE%	AG	mAG
Tune	5.24	13.50	1.01	1.22
EE	23.71	29.34	6.18	8.32
VF	14.01	21.37	3.90	7.31

(StarNet050, CIFAR100, 14)

	E%	mE%	AG	mAG
Tune	4.71	10.94	0.98	2.18
EE	10.17	22.96	3.12	5.01
VF	6.63	14.45	2.06	3.72

(TCResNet14, SCKWS, 25)

	E%	mE%	AG	mAG
Tune	3.48	15.55	0.22	0.60
EE	24.17	36.75	2.05	3.60
VF	36.08	40.03	2.68	3.65

Fig. 5. Ablation studies results by removing finetuning, early-exit, and DVFS, respectively. The test results of valid continuous inference, where the time constraint T^c is not violated, are presented. Each subplot corresponds to a different combination of (model, dataset, T^c(s)). The x-axes represent the energy consumption in joules (J), while the y-axes denote the accuracy of the model in percentage (%). E%, mE%, AG, and mAG brought by tuning, early-exit, and DVFS are reported.

B. Results and Analysis

Fig. 3 presents a comparison of DE^{2}R with other techniques. Each data point shown in the results is time-valid, meaning that the continuous inference task is completed within the time constraint. Energy reduction in percentage, E% and mE%, are reported, representing the average and maximum energy reduction in percentage (%) compared to the second-best method under the same accuracy. On the other hand, we also report the Accuracy Gain, AG and mAG, which are the average and maximum accuracy gain in percentage (%) compared to the second-best method under the same energy consumption.

It can be observed that under various settings, our method generally outperforms the compared approaches, namely BranchyNet [21], DTRL [2] and PredictiveExit [13], by achieving up to and 22.03% E% and 3.23% AG. For MobileNetV2 with three exits, the largest energy reduction is observed on CIFAR100 at $T^c = 30$, where E% reaches 11.26% and mE% reaches 18.59%. The largest AG occurs on CIFAR10 at $T^c = 40$, with AG of 3.23% and mAG of 10.95%. For StarNet050 with four exits, the maximum energy reduction is achieved on CIFAR10 at $T^c = 20$, where E% is 22.03% and mE% reaches 33.03%. The highest AG is observed on CIFAR100 at $T^c = 20$, where AG reaches 1.70% and mAG is 4.95%. For TCResNet14 with two exits, the largest energy reduction is found on SCKWS at $T^c = 25$, where E% reaches 12.59% and mE% is 28.44%. The best AG on GSCD is at $T^c = 14$, where AG reaches 2.01% and mAG is 5.09%.

We also present the action statistics of the agent for the setting (MobileNetV2, CIFAR10, 30) in Fig. 4, which reflects the proportion of actions taken by the agent when different weight vectors $\boldsymbol{w}$ are assigned to the reward vector. The upper half shows action distributions at exit points, while the lower half depicts those for DVFS settings. Each bar represents the aggregated action distribution for specific accuracy levels. It confirms the general assumption that, under the same time constraint, as accuracy increases, the model tends to exit from deeper exit points and use higher V-F settings to accelerate inference, which also leads to higher energy consumption. However, exceptions occur, such as between accuracy 80.28% and 81.92%, where utilization of shallower exit (Exit 1), in combination with deeper exit (Exit 3), leads to higher accuracy compared to more moderate points (Exit 2) used. This suggests why heuristics based on the above-mentioned general assumption may achieve suboptimal results.

We perform ablation studies to assess the contribution of each component of DE^{2}R, as shown in Fig. 5. We compare DE^{2}R with three variations: one without tuning, one without the early-exit mechanism, and one without DVFS. It can be observed that on MobileNetV2 with CIFAR10 at $T^c = 30$, finetuning offers the largest E% of 16.34% and mE% of 34.45%, as well as the largest AG of 2.01% and mAG of 5.54%. The early-exit mechanism provides the largest energy reduction on TCResNet14 with SCKWS at $T^c = 25$ with 24.17% E% and 36.75% mE%, and the highest AG on MobileNetV2 with CIFAR10 at $T^c = 40$ with 9.29% AG and 14.11% mAG. Specifically, under the setting of (TCResNet14, GSCD, 18), removing early-exit results in single valid data point due to the stringent time constraint. Finally, V-F adjustment demonstrates the largest energy reduction on TCResNet14 with SCKWS at $T^c = 25$ with 36.08% E% and 40.03% mE%, and the highest AG on TCResNet14 with GSCD at $T^c = 18$ with 7.12% AG and 10.34% mAG. Overall, the results indicate that each component contributes to the effectiveness of the method.

TABLE II
HYPER-PARAMETERS FOR CROSS-TRAINING OF DE^{2}R

Symbol	Value	Description
N^{total}	100	Total number of inputs of each task
b	8	Mini-batch size
γ	0.99	Discount factor
ϵ	0.1	Policy update clipping parameter
lr_{train}	1e-4	Learning rate for training agent
lr_{tune}	5e-6	Learning rate for tuning the early-exit net
l_{vh}	1	Number of hidden layers in value net
$neuron_{vh}$	32	Number of hidden neurons in value net
l_{ph}	1	Number of hidden layers in policy net
$neuron_{ph}$	64	Number of hidden neurons in policy net

VII. CONCLUSION

In this paper, we propose DE^{2}R, a novel RL-based scheduling method that unifies DVFS and early-exit neural networks to optimize energy consumption and inference accuracy in continuous inference scenarios. We design a cross-training mechanism to fine-tune the early-exit network alongside RL agent policy updates, to further improve the inference accuracy. Experimental results show DE^{2}R outperforms contemporary methods with verified robustness and adaptability.

References

[1] ASR-SCKwsptSC: A Scripted Chinese Keyword Spotting Speech Corpus. https://magichub.com/datasets/mandarin-chinese-scripted-speech-corpus-keyword-spotting-2/ (2024), accessed: 12/09/2024

[2] Basaklar, T., Goksoy, A.A., Krishnakumar, A., Gumussoy, S., Ogras, U.Y.: Dtrl: decision tree-based multi-objective reinforcement learning for runtime task scheduling in domain-specific system-on-chips. ACM Transactions on Embedded Computing Systems **22**(5s), 1–22 (2023)

[3] Chen, F., Yu, H., Jiang, W., Ha, Y.: Quality optimization of adaptive applications via deep reinforcement learning in energy harvesting edge devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **41**(11), 4873–4886 (2022)

[4] Choi, S., Seo, S., Shin, B., Byun, H., Kersner, M., Kim, B., Kim, D., Ha, S.: Temporal Convolution for Real-Time Keyword Spotting on Mobile Devices. In: Proc. Interspeech 2019. pp. 3372–3376 (2019)

[5] Han, Y., Huang, G., Song, S., Yang, L., Wang, H., Wang, Y.: Dynamic neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **44**(11), 7436–7456 (2021)

[6] He, Y., Xiao, L.: Structured pruning for deep convolutional neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence (2023)

[7] Jeon, J.Y., Nguyen, X.T., Ryu, S., Lee, H.J.: Usdn: A unified sample-wise dynamic network with mixed-precision and early-exit. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 646–654 (2024)

[8] Jie, Z., Sun, P., Li, X., Feng, J., Liu, W.: Anytime recognition with routing convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence **43**(6), 1875–1886 (2019)

[9] Jo, J., Kim, G., Kim, S., Park, J.: Locoexnet: Low-cost early exit network for energy efficient cnn accelerator design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **42**(12), 4909–4921 (2023)

[10] Ju, W., Bao, W., Ge, L., Yuan, D.: Dynamic early exit scheduling for deep neural network inference through contextual bandits. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. pp. 823–832 (2021)

[11] Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)

[12] Li, J., Jiang, W., He, Y., Yang, Q., Gao, A., Ha, Y., Özcan, E., Bai, R., Cui, T., Yu, H.: Fidrl: Flexible invocation-based deep reinforcement learning for dvfs scheduling in embedded systems. IEEE Transactions on Computers (2024)

[13] Li, X., Lou, C., Chen, Y., Zhu, Z., Shen, Y., Ma, Y., Zou, A.: Predictive exit: Prediction of fine-grained early exits for computation-and energy-efficient inference. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 37, pp. 8657–8665 (2023)

[14] Lu, H., Herman, D., Yu, Y.: Multi-objective reinforcement learning: Convexity, stationarity and pareto optimality. In: The Eleventh International Conference on Learning Representations (2022)

[15] Ma, X., Dai, X., Bai, Y., Wang, Y., Fu, Y.: Rewrite the stars. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5694–5703 (2024)

[16] Passalis, N., Raitoharju, J., Tefas, A., Gabbouj, M.: Efficient adaptive inference for deep convolutional neural networks using hierarchical early exits. Pattern Recognition **105**, 107346 (2020)

[17] Rokh, B., Azarpeyvand, A., Khanteymoori, A.: A comprehensive survey on model quantization for deep neural networks in image classification. ACM Transactions on Intelligent Systems and Technology **14**(6), 1–50 (2023)

[18] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018)

[19] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

[20] Sepehri, Y., Pad, P., Yüzügüler, A.C., Frossard, P., Dunbar, L.A.: Hierarchical training of deep neural networks using early exiting. IEEE Transactions on Neural Networks and Learning Systems (2024)

[21] Teerapittayanon, S., McDanel, B., Kung, H.T.: Branchynet: Fast inference via early exiting from deep neural networks. In: 2016 23rd international conference on pattern recognition (ICPR). pp. 2464–2469. IEEE (2016)

[22] Warden, P.: Speech commands: A dataset for limited-vocabulary speech recognition. arXiv preprint arXiv:1804.03209 (2018)

[23] Wołczyk, M., Wójcik, B., Bałazy, K., Podolak, I.T., Tabor, J., Śmieja, M., Trzcinski, T.: Zero time waste: Recycling predictions in early exit neural networks. Advances in Neural Information Processing Systems **34**, 2516–2528 (2021)

[24] Yu, H., Ha, Y., Wang, J.: Quality optimization of resilient applications under temperature constraints. In: Proceedings of the Computing Frontiers Conference. pp. 9–16 (2017)

[25] Zhang, Z., Zhao, Y., Li, H., Lin, C., Liu, J.: Dvfo: Learning-based dvfs for energy-efficient edge-cloud collaborative inference. IEEE Transactions on Mobile Computing (2024)

[26] Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., Wei, F.: Bert loses patience: Fast and robust inference with early exit. Advances in Neural Information Processing Systems **33**, 18330–18341 (2020)