DART: Input-Difficulty-AwaRe Adaptive Threshold for Early-Exit DNNs

Parth Patne[1], Mahdi Taheri[1,2], Christian Herglotz[1], Maksim Jenihhin[2], Milos Krstic[3], and Michael Hübner[1]

[1]Brandenburg Technical University, Cottbus, Germany
[2]Tallinn University of Technology, Tallinn, Estonia
[3]Leibniz Institute for High Performance Microelectronics

***Abstract*—Early-exit deep neural networks enable adaptive inference by terminating computation when sufficient confidence is achieved, reducing cost for edge AI accelerators in resource-constrained settings. Existing methods, however, rely on suboptimal exit policies, ignore input difficulty, and optimize thresholds independently. This paper introduces DART (Input-Difficulty-Aware Adaptive Threshold), a framework that overcomes these limitations. DART introduces three key innovations: (1) a lightweight difficulty estimation module that quantifies input complexity with minimal computational overhead, (2) a joint exit policy optimization algorithm based on dynamic programming, and (3) an adaptive coefficient management system. Experiments on diverse DNN benchmarks (AlexNet, ResNet-18, VGG-16) demonstrate that DART achieves up to 3.3× speedup, 5.1× lower energy, and up to 42% lower average power compared to static networks, while preserving competitive accuracy. Extending DART to Vision Transformers (LeViT) yields power (5.0×) and execution-time (3.6×) gains but also accuracy loss (up to 17 percent), underscoring the need for transformer-specific early-exit mechanisms. We further introduce the Difficulty-Aware Efficiency Score (DAES), a novel multi-objective metric, under which DART achieves up to a 14.8× improvement over baselines, highlighting superior accuracy, efficiency, and robustness trade-offs.**

***Index Terms*—Deep Neural Networks, Dynamic Neural Networks, Edge AI, Hardware Accelerators, Efficient Computing**

I. Introduction

Dynamic Deep Neural Networks (D2NNs) address the inefficiency of traditional static inference, where every input traverses the full network despite high energy consumption and latency, by adapting computation to input characteristics, making them more suitable for resource-constrained edge accelerators [1]. Early exit networks, introduced first by BranchyNet [2], integrate auxiliary classifiers at intermediate layers, allowing confident predictions to exit early. This approach demonstrated that many inputs require only shallow processing, enabling significant savings. However, reliance on fixed confidence thresholds and independent exit optimization limits adaptability during real-time execution.

Subsequent work has improved exit policies. Zhou et al. [3] and Taheri et al. [4] employed reinforcement learning to optimize exits, while Xin et al. [5] introduced DeeBERT for transformer acceleration. RACENet [6] incorporates class-awareness through adaptive normalization but relies on computationally expensive MLPs at every layer, negating efficiency gains on constrained edge devices.

Despite all efforts in the literature, three critical limitations persist. First, state-of-the-art approaches [7], [8] optimize exit thresholds independently, ignoring their interdependencies and yielding suboptimal routing policies. Second, the estimation of input complexity is computationally expensive [9], [10] or insufficiently representative of neural processing needs [11], limiting the real-time deployment. Third, current systems adopt static exit policies learned during training, leaving them vulnerable to distribution shifts and operational variability, with little work exploring online adaptation [12], [13] in different contexts that need to be studied in the D2NN domain.

To overcome these limitations, this paper introduces **DART** (Input-Difficulty-Aware Adaptive Threshold), a unified framework featuring: (1) **Difficulty-aware input processing** for lightweight real-time complexity estimation; (2) **Joint exit threshold optimization** using dynamic programming for globally coherent routing; and (3) **Adaptive coefficient management** with online learning–based updates for continuous refinement during deployment.

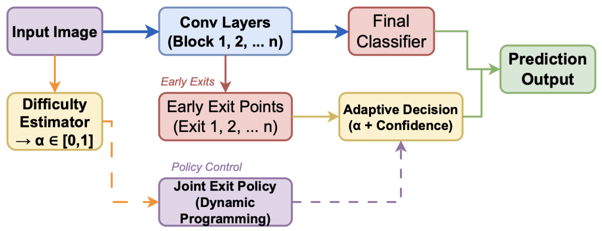
The main contributions of this work are:

- **A unified framework (DART)** that integrates difficulty-aware input processing, joint threshold optimization, and adaptive management to overcome fundamental early-exit limitations;
- **A practical and extensible deployment methodology** that enables real-time execution on edge accelerators while ensuring portability across diverse neural network architectures;
- **Open-source release and comprehensive evaluation** of the fully automated framework on state-of-the-art CNN and vision transformer benchmarks.

The remainder of this paper is structured as follows: Section II presents the proposed methodology; Section III describes the experimental setup and results; Section IV concludes the paper.

II. Methodology

Figure 1 illustrates the system-level integration of DART into an early-exit DNN, showing how difficulty estimation, adaptive thresholding, and coefficient management are realized within the model pipeline. DART implements a systematic approach to dynamic neural network optimization



Input Image
Conv Layers (Block 1, 2, ... n)
Final Classifier
Prediction Output
Difficulty Estimator → α ∈ [0,1]
Early Exits
Early Exit Points (Exit 1, 2, ... n)
Adaptive Decision (α + Confidence)
Policy Control
Joint Exit Policy (Dynamic Programming)

Fig. 1: System-level architecture of the proposed Input-Difficulty-Aware early-exit DNN with DART integration.

through three interconnected components that work synergistically to achieve optimal efficiency–accuracy trade-offs. Within a DART-enabled model, inputs are pre-processed by the lightweight difficulty-estimation module, exit thresholds are dynamically adjusted using the learned policies, and adaptation is performed online during inference. The framework itself provides the methodology and optimization algorithms that enable these runtime modules.

A. Difficulty-Aware Input Processing

The foundation of DART lies in its ability to quantify input complexity through a lightweight preprocessing module. This module combines three complementary metrics to provide a comprehensive difficulty assessment.

1) Multi-Modal Difficulty Metrics: **Edge Density Computation:** Edge density quantifies the structural complexity of input images through gradient analysis. The computation employs Sobel operators in both horizontal and vertical directions:

$$G_x(x,y) = I(x,y) * S_x, \tag{1}$$
$$G_y(x,y) = I(x,y) * S_y, \tag{2}$$
$$G(x,y) = \sqrt{G_x^2(x,y) + G_y^2(x,y)}, \tag{3}$$

where $I(x,y)$ represents the grayscale-converted input image of height H and width W, S_x and S_y are the Sobel kernels [14], and $*$ denotes convolution. The edge density score is computed as:

$$\alpha_{\text{edge}} = \frac{1}{HW}\sum_{x=1}^{H}\sum_{y=1}^{W}\mathbf{1}[G(x,y) > \tau_{\text{edge}}], \tag{4}$$

where $\mathbf{1}[\cdot]$ is the indicator function and τ_{edge} is an adaptive threshold tuned per scene complexity [9].

Pixel Variance Analysis: Spatial variance captures texture complexity and local variations within the input. For an input tensor $X \in \mathbb{R}^{B\times C\times H\times W}$ (where B is batch size, C is channels, H is height, and W is width), the variance is computed across spatial dimensions:

$$\mu_{b,c} = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} X_{b,c,h,w}, \tag{5}$$
$$\alpha_{\text{variance}} = \frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W}(X_{b,c,h,w} - \mu_{b,c})^2, \tag{6}$$

following established texture complexity analyses.

Gradient Complexity Assessment: Second-order spatial variations are captured using Laplacian operators to detect fine-grained patterns:

$$\alpha_{\text{gradient}} = \frac{1}{HW}\sum_{x=1}^{H}\sum_{y=1}^{W}|L(x,y)|, \quad L(x,y) = I(x,y)*K_{\text{laplacian}}, \tag{7}$$

where $K_{\text{laplacian}}$ is the Laplacian kernel, reflecting classic second-order edge detection.

2) Difficulty Score Fusion: The final difficulty score combines all three metrics through a weighted fusion approach:

$$\alpha = w_1 \cdot \alpha_{\text{edge}} + w_2 \cdot \alpha_{\text{variance}} + w_3 \cdot \alpha_{\text{gradient}}, \tag{8}$$

where $\alpha \in [0,1]$ is the final difficulty score, and the weights (w_1, w_2, w_3) are empirically determined via cross-validation using random search optimization [15].

3) Difficulty-Aware Efficiency Score (DAES): To comprehensively evaluate the effectiveness of difficulty-aware routing, we introduce the **Difficulty-Aware Efficiency Score (DAES)**, which combines accuracy, computational efficiency, and robustness to input complexity:

$$\text{DAES} = \frac{\text{Accuracy} \times \text{Speedup} \times \text{Power\_Efficiency}}{1+\alpha} \tag{9}$$

Here, Power_Efficiency(m) is defined in Eq. 22, and $\alpha \in [0,1]$ is the difficulty score computed per input using Eq. 8.

Difficulty_Score $\in [0,1]$ is computed for each test input using our multi-modal difficulty estimation framework (Eq. 8). This ensures a fair complexity-aware evaluation across all methods, including static and non-difficulty-aware baselines, without introducing additional scaling factors.

The DAES metric captures the fundamental trade-off in early-exit networks by jointly considering accuracy, efficiency, and robustness to input complexity. Higher DAES scores indicate models that achieve more balanced performance across heterogeneous input conditions, making the metric particularly relevant for real-world deployment.

B. Joint Exit Policy Optimization

The core innovation of DART lies in its joint optimization approach that considers all exit points simultaneously, formulated as a global optimization problem to maximize overall efficiency and accuracy trade-offs.

1) Problem Formulation: Given a neural network with N exits, the optimization objective seeks threshold values $\boldsymbol{\tau} = [\tau_1, \tau_2, \ldots, \tau_{N-1}]$ that maximize:

$$\mathcal{J}(\boldsymbol{\tau}) = \sum_{i=1}^{N}\pi_i(\boldsymbol{\tau})[A_i - \beta_{\text{opt}} \cdot C_i], \tag{10}$$

where $\pi_i(\boldsymbol{\tau})$ is the probability of exiting at layer i, A_i is the accuracy achieved when exiting at layer i, C_i is the computational cost incurred up to exit i, and $\beta_{\text{opt}} \in [0,1]$ is a trade-off parameter that balances accuracy preservation against computational efficiency gains, following dynamic programming principles [16].

2) Dynamic Programming Solution: We employ a value iteration algorithm over state representations $s = (\text{exit\_index}, \alpha_{\text{bin}}, \text{confidence}_{\text{bin}})$ to learn optimal exit policies. The Q-value update combines immediate rewards and future expected returns:

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)\, V(s'), \tag{11}$$

where $R(s,a)$ balances accuracy and cost, γ is the discount factor, and transitions follow the MDP framework [17].

3) Threshold Calibration: We generate candidate thresholds using quantiles of the confidence distributions:

$$\tau_i^{\text{candidate}} = \text{quantile}(\mathcal{C}_i, q), \quad q \in \{0.1, 0.2, \ldots, 0.9\}, \tag{12}$$

where $\mathcal{C}_i$ is the confidence distribution at exit i, following practices in early-exit tuning [3].

C. Adaptive Coefficient Management

The adaptive management system continuously refines exit policies through multi-scale strategies as follows.

1) Multi-Strategy Adaptation Framework: **Temporal Adaptive Strategy:** Coefficients evolve based on recent performance via exponential decay:

$$c_t = \alpha_{\text{decay}} \cdot c_{t-1} + (1 - \alpha_{\text{decay}}) \cdot f(\text{performance}_t), \tag{13}$$

where $\alpha_{\text{decay}} \in [0,1]$ is the decay factor, following online learning formulations [18].

Class-Aware Adaptation: Class-specific coefficients update based on per-class performance:

$$c_{\text{class}}^{(t+1)} = c_{\text{class}}^{(t)} + \eta \cdot (A_{\text{target}} - A_{\text{class}}^{(t)}), \tag{14}$$

where A_{target} is the desired accuracy (e.g., 0.85), $A_{\text{class}}^{(t)}$ is the current accuracy for the specific class (e.g., "car" or "ship" in CIFAR-10), and η is the adaptation rate, inspired by meta-learning updates [12]. During deployment, we use the model's high-confidence predictions as pseudo-labels to update class statistics in the absence of ground truth.

2) Adaptive Selection and Tracking: We maintain running statistics over a sliding window of the most recent $w = 1000$ inferences (per exit and per class): accuracy, confidence distributions, and compute (time/energy). These statistics drive small periodic updates of the adaptive coefficients and thresholds.

When multiple adaptation strategies (e.g., alternative coefficient sets) are available, we select the next strategy using the UCB1 rule to balance exploration and exploitation. UCB1 is a multi-armed bandit algorithm that intelligently chooses between trying new strategies and using proven successful ones, ensuring DART continuously learns optimal coefficient management, as formulated in the following equation:

$$\text{UCB}_i(t) = \bar{r}_i(t) + \sqrt{\frac{2 \ln t}{n_i(t)}}, \tag{15}$$

where $\bar{r}_i(t)$ is the windowed reward estimate (accuracy–cost trade-off from Eq. 10) and $n_i(t)$ counts selections of strategy i. If UCB selection is disabled, the system reduces to deterministic threshold adaptation driven by the same sliding-window statistics.

D. Framework Extensibility to Transformers

To demonstrate the adaptability of our CNN-focused framework, we extended DART to vision transformers. For LeViT transformers, the framework adapts to token-based representations:

$$\text{ExitBlock}_{\text{ViT}}(T) = \text{MLP}(\text{LayerNorm}(\text{GlobalPool}(T))) \tag{16}$$

where $T \in \mathbb{R}^{B \times N \times D}$ represents the token sequence with batch size B, N tokens, and D dimensions.

Token-Level Difficulty Estimation: For transformers, we maintain the same difficulty estimation approach as CNNs, computing α from the input image before tokenization to ensure consistent difficulty assessment across architectures:

$$\alpha_{\text{token}} = w_1 \cdot \alpha_{\text{edge}} + w_2 \cdot \alpha_{\text{variance}} + w_3 \cdot \alpha_{\text{gradient}} \tag{17}$$

where the difficulty components are computed from the input image using Equations 4, 6, and 7 before tokenization, with weights (w_1, w_2, w_3).

E. Training and Inference Pipeline

The training process integrates all components through unified optimization that simultaneously learns network parameters and exit policies.

1) Multi-Exit Loss Function: The total loss combines contributions from all exits with progressive weighting:

$$\mathcal{L}_{\text{total}} = \sum_{i=1}^{N} w_i \cdot \mathcal{L}_{\text{CE}}(y, \hat{y}_i) + \lambda \cdot \mathcal{L}_{\text{policy}} \tag{18}$$

where $w_i = \frac{i}{N}$ emphasizes later exits. The term $\mathcal{L}_{\text{CE}}$ represents the standard Cross-Entropy loss between the true labels y and the prediction $\hat{y}_i$ from each exit. The term $\mathcal{L}_{\text{policy}}$ is a regularization loss that encourages an efficient exit distribution by penalizing overuse of later exits.

2) Inference Optimization: During inference, the framework applies the learned exit policy while adapting thresholds to the input difficulty. We *increase* the confidence requirement for difficult inputs to avoid premature early exits. Let $\alpha \in [0,1]$ denote the difficulty score of the current input (higher means harder), $\boldsymbol{\tau} \in [0,1]^{N-1}$ the learned base thresholds, and $\boldsymbol{c} \in \mathbb{R}_{+}^{N-1}$ the learned per-exit coefficients. We first apply the coefficients element-wise,

$$\boldsymbol{\tau}_{\text{adapted}} = \boldsymbol{c} \odot \boldsymbol{\tau},$$

and then form the difficulty-aware effective threshold at exit i as

$$\tau_i' = (\tau_{\text{adapted}})_i + \beta_{\text{diff}} \cdot \alpha, \tag{19}$$

followed by clamping to $[0,1]$. Thus, easy inputs ($\alpha \approx 0$) retain near-baseline thresholds, while hard inputs ($\alpha \approx 1$) face

Algorithm 1 DART Adaptive Exit Decision Algorithm

```
1:  Input: Sample $x$, learned thresholds $\boldsymbol{\tau}$, coefficients $\boldsymbol{c}$,
    difficulty scale $\beta_{\text{diff}} \geq 0$
2:  Compute difficulty score: $\alpha \leftarrow f_{\text{difficulty}}(x)$
3:  Apply adaptive coefficients: $\boldsymbol{\tau}_{\text{adapted}} \leftarrow \boldsymbol{c} \odot \boldsymbol{\tau}$
4:  for $i = 1$ to $N - 1$ do
5:      Compute exit prediction: $(\hat{y}_i, \text{conf}_i) \leftarrow E_i(h_i)$
6:      Difficulty-aware threshold (Eq. 19): $\tau'_i \leftarrow (\tau_{\text{adapted}})_i + \beta_{\text{diff}} \cdot \alpha$
7:      Clamp to $[0, 1]$: $\tau'_i \leftarrow \min(1, \max(0, \tau'_i))$
8:      if $\text{conf}_i > \tau'_i$ then
9:          return $\hat{y}_i$, exit_index $= i$
10:     end if
11: end for
12: return Final prediction from exit $N$
```

higher thresholds and are more likely to continue to deeper exits. We tune the nonnegative sensitivity parameter $\beta_{\text{diff}} \geq 0$ on a validation set to balance responsiveness and stability.

This pipeline ensures that exit decisions incorporate both learned policies and real-time input characteristics, enabling adaptive inference that responds to input complexity variations.

III. Experimental Evaluation

Experiments were conducted to evaluate DART across accuracy, computational efficiency, and adaptive behavior. All experiments were implemented in PyTorch with CUDA acceleration and executed on NVIDIA A100 GPUs. While experiments utilized A100 GPUs, the reported MACs and energy savings are architecture-agnostic metrics that directly translate to efficiency gains on edge accelerators.

Three representative CNN architectures were selected as primary testbeds: AlexNet (8 layers, 61M parameters), ResNet-18 (18 layers, 11M parameters), and VGG-16 (16 layers, 138M parameters). MNIST and CIFAR-10 were used as two standard datasets for evaluation. To further examine framework generality, experiments were extended to LeViT-128s, a transformer-based model with 7.8M parameters, serving as a proof-of-concept for transformer applications.

The difficulty estimation module used empirically determined weights of $(w_1 = 0.4, w_2 = 0.3, w_3 = 0.3)$ for fusing edge density, pixel variance, and gradient complexity metrics. The difficulty-aware threshold adaptation used $\beta_{\text{diff}} = 0.3$ as the sensitivity parameter. The adaptive coefficient manager employed an exponential decay strategy with $\alpha_{\text{decay}} = 0.95$.

Performance was benchmarked against two baselines, including static models without early exits, and BranchyNet with fixed thresholds [2]. Evaluation metrics included top-1 accuracy, computational cost in multiply–accumulate (MAC) operations, wall-clock inference time, and energy consumption measured via NVIDIA-SMI. In addition, exit distributions were analyzed to assess routing patterns and adaptive behavior.

1) Metric Definitions: All latency, energy, and power measurements were obtained under identical conditions (same batch size, precision, dataloader, and warmup). Each method

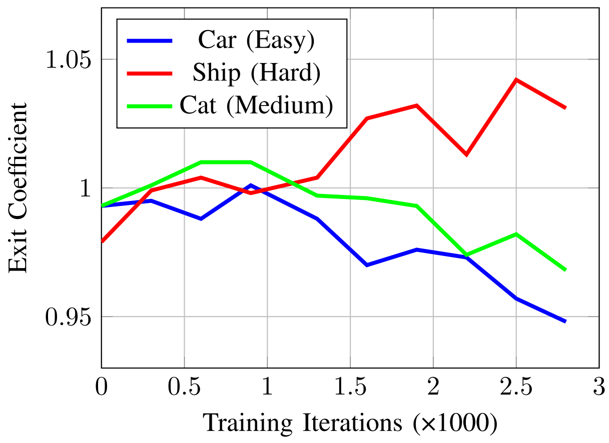


Fig. 2: Evolution of adaptive coefficients during training for three CIFAR-10 classes based on real experimental data.

was executed R times, and the median latency was used to compute derived metrics.

Speedup relative to static baselines was calculated as:

$$\text{Speedup}(m) = \frac{T_{\text{Static}}}{T_m}, \tag{20}$$

where T_m is the per-inference wall-clock time of method m. Average power was defined as:

$$P_m = \frac{E_m}{T_m}, \tag{21}$$

with energy E_m obtained by integrating instantaneous power during inference. Finally, normalized power efficiency was reported as:

$$\text{Power\_Efficiency}(m) = \frac{E_{\text{Static}}}{E_m}, \tag{22}$$

with $\text{Power\_Efficiency}(\text{Static}) = 1.0\times$ by definition.

A. Performance Comparison

Table I summarizes the results across all models and datasets.

DART consistently improves efficiency across CNNs. On CIFAR-10, it achieves a 1.60× speedup on AlexNet (82.86% vs 85.29% accuracy, 2.7× energy reduction), a 2.25× speedup on ResNet-18 (85.35% vs 88.32% accuracy, 2.3× energy reduction), and a 3.33× speedup on VGG-16 (80.20% vs 79.16% accuracy, 3.7× energy reduction). On MNIST, AlexNet with DART reaches 3.0× speedup and 5.1× energy reduction while slightly improving accuracy (99.31% vs 98.97%).

For LeViT transformers, DART achieves 2.53–3.58× speedups but with accuracy reductions of up to 17 percentage points, indicating that CNN-oriented early-exit strategies are not directly transferable to attention-based models.

B. Difficulty-Aware Performance

Observing CIFAR-10 inputs, it is evident that they exhibit moderate complexity with a mean difficulty score of $\alpha \approx 0.85$, consistent across models, confirming that difficulty reflects dataset characteristics rather than architectural bias.

TABLE I: Performance and Difficulty-Aware Analysis (MNIST $\alpha = 0.76$; CIFAR-10 $\alpha = 0.85$).

Architecture	Method	Acc.(%)	Time(ms)	Energy(mJ)	Power(W)	Speedup	Power Eff.	DAES
MNIST Results								
AlexNet	Static	98.97	0.09	5.90	65.56	1.0×	1.00	0.562
	BranchyNet	99.13	0.08	5.27	65.88	1.13×	1.12	0.713
	RL-Agent	99.49	0.06	2.29	38.20	1.50×	2.57	2.179
	DART	99.31	0.03	1.15	38.33	3.0×	5.13	8.684
CIFAR-10 Results (CNNs)								
AlexNet	Static	85.29	0.08	5.18	64.75	1.0×	1.00	0.461
	BranchyNet	83.15	0.07	4.57	65.29	1.14×	1.13	0.579
	RL-Agent	84.42	0.05	1.84	36.73	1.60×	2.82	1.888
	DART	82.86	0.05	1.88	37.60	1.60×	2.76	1.978
ResNet-18	Static	88.32	0.27	17.66	65.41	1.0×	1.00	0.477
	BranchyNet	87.72	0.16	10.43	65.19	1.69×	1.69	1.354
	RL-Agent	87.87	0.14	8.10	57.84	1.93×	2.18	1.998
	DART	85.35	0.12	7.53	62.75	2.25×	2.35	2.439
VGG-16	Static	79.16	0.20	9.41	47.05	1.0×	1.00	0.428
	BranchyNet	81.82	0.11	3.70	46.25	1.82×	2.54	4.63
	RL-Agent	80.89	0.10	3.91	39.13	2.00×	2.40	2.021
	DART	80.20	0.06	2.54	42.33	3.33×	3.71	5.356
CIFAR-10 Results (LeViT Transformers)								
LeViT-128S	Static	95.80	11.55	750.75	65.00	1.0×	1.00	0.518
	DART	81.73	4.56	150.10	32.92	2.53×	5.00	5.588
LeViT-192	Static	96.91	19.94	1296.10	65.00	1.0×	1.00	0.524
	DART	80.33	5.57	259.18	46.53	3.58×	5.00	7.772
LeViT-256	Static	97.28	62.55	4065.75	65.00	1.0×	1.00	0.526
	DART	86.11	18.89	813.30	43.05	3.31×	5.00	7.704

TABLE II: Extensibility Study: LeViT Transformer Performance Analysis on CIFAR-10

Model	Method	Acc. (%)	MACs (M)	Time (ms)	Speedup (×)
LeViT-128S	Static	95.80	282.1	11.55	1.00×
	DART	81.73	56.4	4.56	2.53×
LeViT-192	Static	96.91	601.1	19.94	1.00×
	DART	80.33	120.2	5.57	3.58×
LeViT-256	Static	97.28	1053.3	62.55	1.00×
	DART	86.11	210.7	18.89	3.31×

Table I reports DAES values alongside standard performance metrics. Three insights emerge. First, CNNs benefit substantially: for example, VGG-16 improves from 0.428 (static) to 5.356 (DART), a 12.5× gain. Second, transformers also exhibit improvements: LeViT-128S improves from 0.518 to 5.588 (10.8×), LeViT-192 from 0.524 to 7.772 (14.8x), and LeViT-256 from 0.526 to 7.704 (14.6×), showing that DART's principles generalize beyond CNNs. Third, across all architectures, DAES improvements confirm that DART exploits input complexity effectively to achieve favorable efficiency–accuracy trade-offs.

To validate the efficiency of DART's difficulty-estimation module against the state of the art, the computational overhead is compared against RACENet [6] using FLOPs and latency. Only the control mechanisms responsible for dynamic behavior are considered to ensure a fair overhead comparison, i.e., DART's input-difficulty estimator and RACENet's class-aware adaptive normalization. Latency values were measured on an NVIDIA GPU with a batch size of 128, averaged over 5,000 runs to capture per-sample throughput. DART's `DifficultyEstimator` adds 78.9K FLOPs for lightweight input analysis, whereas RACENet's adaptive normalization requires a dedicated MLP at every layer, contributing 716,912 additional parameters and 3.96M FLOPs. This results in RACENet incurring **50.3**× higher compute overhead and a substantially larger memory footprint than DART, clearly highlighting its inefficiency for deployment under resource constraints

C. Adaptive Behavior

Figure 2 illustrates the evolution of exit coefficients during training. The adaptive system learns class-specific strategies: for easy classes (car), coefficients decrease from 0.99 to 0.95, enabling more aggressive early exits. For hard classes (ship), coefficients increase from 0.98 to 1.05, resulting in more conservative exits. Medium classes (cat) show intermediate evolution (0.99 to 0.97). This demonstrates the framework's ability to adapt policies dynamically based on real data distributions.

D. CNN Architecture Analysis

A detailed analysis of CNNs highlights architecture-dependent benefits. AlexNet achieves the highest power savings (42%), while ResNet-18 and VGG-16 show distinct accuracy efficiency trade-offs, underscoring the importance of architecture-aware tuning. The results confirm that DART's joint optimization converges to stable exit distributions, effectively balancing accuracy and efficiency across multiple exits while learning meaningful exit strategies.

E. ViT Architecture Analysis

The extensibility study with LeViT models provides insight into applying DART to non-CNN architectures. While efficiency gains are consistent (2.5–3.6× speedups, substantial energy reduction), accuracy drops remain noticeable (up to 17 points). This limitation highlights fundamental differences between CNN and transformer representations: early transformer layers primarily capture token positioning and structural cues rather than semantic features, making early exits more detrimental than in CNNs. These results point to the need for transformer-specific early-exit mechanisms, such as attention-aware difficulty metrics, token-level exit strategies, and specialized calibration.

Overall, the experimental results confirm that DART achieves consistent improvements across CNNs, with speedups of up to 3.33× and energy savings exceeding 5× while maintaining competitive accuracy. The framework adapts effectively to varying input complexities, learns class-dependent strategies in real time. The findings demonstrate the practicality of DART as a unified framework for dynamic neural network optimization on resource-constrained edge AI platforms.

IV. Conclusion

This paper introduces **DART** (Input-Difficulty-Aware Adaptive Threshold), a framework that enhances early-exit neural networks by tackling suboptimal exit policies, missing input-difficulty awareness, and independent threshold optimization. DART integrates three innovations: (1) a lightweight difficulty estimation module with minimal overhead, (2) a joint exit policy optimization via dynamic programming, and (3) an adaptive coefficient management system. Experiments on AlexNet, ResNet-18, and VGG-16 show up to **3.3**× speedup, **5.1**× lower energy, and **42%** lower average power versus static networks, with competitive accuracy. Extending DART to Vision Transformers (LeViT) yields power (5.0×) and execution-time (3.6×) gains but also accuracy loss (up to 17 percent), underscoring the need for transformer-specific early-exit mechanisms. Finally, the proposed Difficulty-Aware Efficiency Score (DAES) demonstrates up to **14.8**× improvement over baselines, capturing DART's superior accuracy–efficiency–robustness trade-offs.

Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS", EU Grant Project 101160182 "TAICHIP" and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID "458578717".

References

[1] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7436–7456, 2022.

[2] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognition (ICPR)*, Dec. 2016, pp. 2464–2469.

[3] W. Zhou, X. Xiong, F. Ge, Z. Mao, and C. Wu, "Bert loses patience: Fast and robust inference with early exit," in *Proc. 34th Conf. Neural Information Processing Systems (NeurIPS)*, Dec. 2020, pp. 19 330–19 341.

[4] M. Taheri, P. Patne, N. Cherezova, A. Mahani, C. Herglotz, and M. Jenihhin, "Rl-agent-based early-exit dnn architecture search framework," in *2025 IEEE 28th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2025, pp. 145–148.

[5] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, "Deebert: Dynamic early exiting for accelerating bert inference," in *Proc. 58th Annual Meeting Association Computational Linguistics*, Jul. 2020, pp. 2246–2251.

[6] M. Ayyat, M. Osman, and T. Nadeem, "Racenet: Real-time adaptive class-aware early-exit networks for edge devices," in *2025 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2025, pp. 152–158.

[7] J. Wang, B. Li, and G. L. Zhang, "Early-exit with class exclusion for efficient inference of neural networks," in *Design, Automation & Test in Europe Conference (DATE)*, 2024, pp. 1–6.

[8] B. Wójcik *et al.*, "Zero time waste in pre-trained early exit neural networks," *Neural Networks*, vol. 168, pp. 593–604, 2023.

[9] E. Peli, "Contrast in complex images," *Journal of the Optical Society of America A*, vol. 7, no. 10, pp. 2032–2040, 1990.

[10] A. Forsythe, M. Mulhern, and M. Sawey, "Confounds in pictorial sets: The role of complexity and familiarity in basic-level picture processing," *Behavior Research Methods*, vol. 40, no. 1, pp. 116–129, 2008.

[11] Y. Lee, J. Lee, S. J. Hwang, E. Yang, and S. Choi, "Neural complexity measures," in *Advances in Neural Information Processing Systems*, ser. NeurIPS, vol. 34, Dec. 2020, pp. 4372–4382.

[12] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Machine Learning (ICML)*, Aug. 2017, pp. 1126–1135.

[13] J. T. Ash and R. P. Adams, "On warm-starting neural network training," in *Proc. 33rd Conf. Neural Information Processing Systems (NeurIPS)*, Dec. 2020, pp. 3884–3894.

[14] I. Sobel and G. Feldman, "A 3×3 isotropic gradient operator for image processing," Stanford Artificial Intelligence Project, Tech. Rep., 1968.

[15] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[16] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[17] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[18] S. Shalev-Shwartz, "Online learning and online convex optimization," *Foundations and Trends in Machine Learning*, vol. 4, no. 2, pp. 107–194, 2012.