ADAPT: Automated Decision-flow for Adaptive Progressive Inference on Sensor Devices

Jared Macshane
jmacshan@uci.edu
University of California, Irvine
Irvine, California, USA

Yuqiao Li
yuqiaol@uci.edu
University of California, Irvine
Irvine, California, USA

Nalini Venkatasubramanian
nalini@ics.uci.edu
University of California, Irvine
Irvine, California, USA

Abstract

The deployment of deep learning models for real-time image classification on resource-constrained sensor devices presents significant challenges. These devices face strict limitations in computational power, energy capacity, and memory resources, making it difficult to achieve both high accuracy and low latency. Current approaches either compromise model performance through compression or incur substantial overhead by offloading computation to remote servers. We introduce a novel distributed progressive inference platform that addresses these limitations by dynamically balancing local and remote computation. Our system employs reinforcement learning to make intelligent decisions about when and where to perform inference. Experimental results across multiple standard datasets demonstrate that our approach achieves up to 3% higher accuracy while reducing network traffic and preserving battery life compared to existing methods.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; **Cooperation and coordination**; **Mobile agents**; • **Computer systems organization** → **Sensor networks**; • **Applied computing** → **Cartography**; **Agriculture**.

Keywords

edge computing, sensor networks, resource-constrained devices, split computing, progressive inference, early-exit networks, deep reinforcement learning, adaptive offloading

ACM Reference Format:
Jared Macshane, Yuqiao Li, and Nalini Venkatasubramanian. 2025. ADAPT: Automated Decision-flow for Adaptive Progressive Inference on Sensor Devices. In *Middleware for Autonomous AIoT Systems in the Computing Continuum (MAIoT '25), December 15–19, 2025, Nashville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3774901.3778065

1 Introduction

Edge computing and sensor devices are enabling robust visual data processing across diverse application domains. Resource-constrained devices like autonomous vehicles, environmental monitoring drones, and IoT surveillance systems increasingly leverage convolutional neural networks (CNNs) for sophisticated computer vision tasks [13, 16, 18]. However, deploying these computationally intensive deep learning models on embedded hardware with limited processing capabilities, memory constraints, and power budgets introduces significant technical challenges [14]. The growing need to process data closer to the source while maintaining model accuracy and real-time performance requirements necessitates novel approaches for efficient edge inference.

Time-critical sensor applications face strict latency requirements, for instance, environmental monitoring must process data streams in real-time to detect critical events such as wildfire detection, obstacle avoidance, and human detection. The key limitations for local sensor inference are computational power, battery life, and memory capacity. Modern CNNs deployed on sensor nodes require billions of operations per inference, overwhelming embedded processors designed for low-power operation. Local neural network inference rapidly depletes the limited battery reserves.

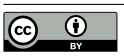
Although model compression techniques such as pruning and quantization can enable local inference, they often degrade accuracy and limit model capabilities - for example, quantized ResNet-50 [3] suffers a 2% accuracy drop [2, 4]. Pure cloud offloading introduces unacceptable communication delays and reliability concerns for sensor networks operating in remote environments [19].

Split computing is a paradigm which allows for a combination of local and remote computation by splitting a neural network across multiple devices [5, 14]. Early exits are a small set of inference layers attached to the intermediate layers of a larger neural network that can terminate inference early based on a predetermined confidence threshold [15, 23].

Recent work on split computing and early exits demonstrate the potential benefits of approaches that can reduce computational load, latency, and model 'overthinking' [6, 10, 14, 23]. However, early exit strategies typically rely on predetermined static thresholds. This motivates our focus on intelligent local inference strategies that can maximize accuracy within sensor resource constraints while selectively leveraging remote computation when beneficial.

We present ADAPT, a distributed progressive inference system for deploying deep learning on resource-constrained devices. Our system combines model partitioning with reinforcement learning to optimize resource usage across sensor nodes and server infrastructure. By implementing multiple inference exit points and dynamically selecting exit points, ADAPT reduces computational and energy costs while maintaining accuracy. The system's key innovations include:

- A novel reinforcement learning agent that makes real-time decisions about early exit points and server offloading based on battery levels, network conditions, and inference confidence



MAIoT '25, Nashville, TN, USA

ACM ISBN 979-8-4007-2304-9/25/12
https://doi.org/10.1145/3774901.3778065

- A flexible progressive inference architecture that enables dynamic partitioning of neural networks between resource-constrained devices and server infrastructure
- A comprehensive resource management framework that optimizes for multiple objectives including accuracy, latency, and energy consumption

Through evaluation on CIFAR-10, Tiny ImageNet, and Visual Wake Words datasets[1, 7, 11], on a custom built simulation platform, with parameters tuned for realistic sensor device constraints, across a range of model sizes and complexities, we demonstrate that ADAPT achieves improvement in energy efficiency compared to baseline approaches while maintaining competitive accuracy. Our results validate the effectiveness of combining progressive inference with reinforcement learning for enabling efficient deep learning on resource-limited sensor devices.

2 Landscape of Confidence Thresholds

In order to motivate our approach, we first examine the limitations of traditional early exit strategies in neural networks. Prior work has predominantly relied on static confidence thresholds to determine exit decisions [14]. Through empirical analysis of the relationship between accuracy and latency across different confidence thresholds, we observe subtle but evident non-uniform patterns in the decision landscape. As illustrated in Figure 1, certain

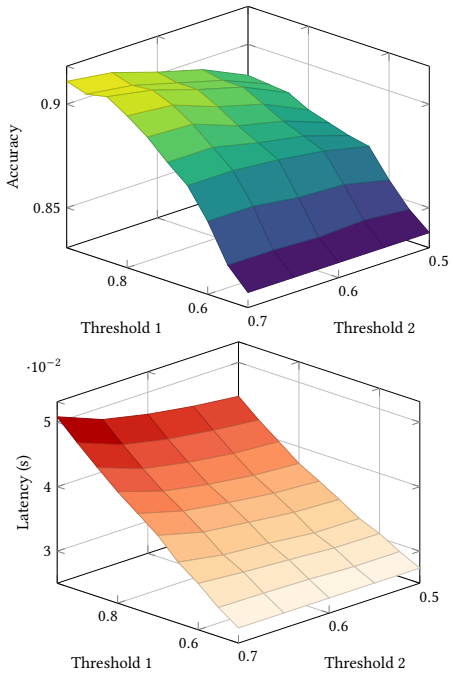


Figure 1: Accuracy and Latency for Confidence Thresholds 1 & 2 (Threshold 3 fixed at 0.75)

threshold combinations can achieve higher classification accuracy while maintaining equivalent latency. This relationship between thresholds and performance metrics motivated our investigation of reinforcement learning as an adaptive control mechanism. By combining a lightweight CNN feature extractor with the ability to analyze exit logits (probability vectors), we hypothesize that an RL agent can effectively learn optimal exit policies that maximize accuracy within given resource constraints.

3 ADAPT: System Design

Our system implements a two-tier split computing architecture optimized for resource-constrained sensor devices. The architecture consists of:

- **Sensor Devices**: Resource-constrained nodes equipped with cameras and basic computing capabilities. These devices perform initial feature extraction and early-exit inference.
- **Server Layer**: A remote computing infrastructure (edge or cloud) that provides additional computational resources for inference tasks when needed.

The split computing approach enables flexible partitioning of the neural network across these two tiers. Early layers are deployed on sensor devices, allowing for immediate feature extraction and potential early exit decisions. More complex layers can be offloaded to the server layer based on resource availability and accuracy requirements.

- **Battery Constraints**: By performing early-exit inference locally, devices can avoid energy-intensive data transmission when confidence is high.
- **Bandwidth Limitations**: Progressive feature extraction reduces the amount of data that needs to be transmitted to the server.

3.1 Progressive Inference Architecture

We introduce the concept of a block and early exit. A block is a coherent set of layers: convolutional, pooling, etc. that we can view as extracting features at different levels of abstraction [3, 8, 12, 20, 22].

The total **loss function** $\mathcal{L}$ is defined as a weighted sum of cross-entropy losses from each exit point:

$$\mathcal{L} = \sum_{i=1}^{N} w_i \mathcal{L}_i \tag{1}$$

where N is the number of exit points, w_i is the weight for exit i, and $\mathcal{L}_i$ is the cross-entropy loss at exit i defined as:

$$\mathcal{L}_i = -\sum_{c=1}^{C} y_c \log(p_{i,c}) \tag{2}$$

3.2 Reinforcement Learning Agent

Q-learning is a model-free reinforcement learning algorithm that learns an optimal action-selection policy by estimating action-values (Q-values) through interactions with the environment. The Q-value $Q(s, a)$ represents the expected cumulative reward when taking action a in state s and following the optimal policy thereafter. The Q-learning update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{3}$$

where α is the learning rate, γ is the discount factor, r_t is the immediate reward, and $\max_a Q(s_{t+1}, a)$ is the maximum Q-value for the next state.

Because our action space is a mixture of continuous (images, logits, battery level) and discrete (exit number), we leverage a Deep Q-Network (DQN) to learn the optimal policy. The action space $\mathcal{A}$ consists of three discrete actions:

- a_1: Continue computation to next block
- a_2: Exit and return predicted label
- a_3: Transmit current state to server for inference

The observation space $\mathcal{S}$ is defined as the tuple:

$$s_t = (e_t, b_t, \mathbf{l}_t, \mathbf{x}_t) \tag{4}$$

- e_t is a one-hot encoded vector for the current exit
- $b_t \in [0, 1]$ represents the normalized battery level
- $\mathbf{l}_t \in \mathbb{R}^C$ is the logit vector for C classes
- $\mathbf{x}_t$ is the input image tensor

Algorithm 2 presents our DQN training approach, which incorporates two key mechanisms for stable learning. First, we utilize experience replay as shown in Algorithm 1, which maintains a buffer of past state-action-reward-state transitions. This allows the agent to learn from a diverse set of historical experiences. The buffer is continuously populated with new transitions as the agent interacts with the simulated environment according to Algorithm 1.

Second, we employ a target network architecture where a separate copy of the main Q-network is maintained and updated less frequently. This target network provides stable Q-value estimates for action selection during training, while the main network is updated regularly to learn from new experiences.

Algorithm 1 Experience Collection

Require: Replay buffer capacity N
Initialize replay buffer $\mathcal{D}$ with capacity N
for each episode **do**
  Initialize state s_1
  for $t = 1$ to T **do**
    With probability ϵ, select random action a_t, otherwise select $a_t = \arg\max_a Q(s_t, a)$
    Execute a_t, observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    $s_t \leftarrow s_{t+1}$
  end for
end for

We devise a heuristic **reward function** to guide the training process, balancing the trade-offs between computation, accuracy, and latency. The reward function is formulated as:

$$r_t = \begin{cases} \alpha_c & \text{if } a_t = 0 \text{ (continue)} \\ \alpha_e \cdot (1 - \frac{e_t}{E}) + \beta_e \cdot c_t & \text{if } a_t = 1 \text{ (early exit)} \\ \alpha_l \cdot \log(L_t) + \beta_l \cdot c_t & \text{if } a_t = 2 \text{ (cloud offload)} \end{cases} \tag{5}$$

where $\alpha_c < 0$ is the continue penalty that discourages unnecessary computation, α_e, β_e are early exit weights that incentivize earlier exits while maintaining accuracy, α_l, β_l are cloud offload

Algorithm 2 DQN Training with Experience Replay

Require: Batch size B, discount factor γ, target update interval K
Initialize Q-network Q with random weights θ
Initialize target network Q' with weights $\theta' \leftarrow \theta$
for Until convergence **do**
  Sample minibatch of B transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
  Set $y_j = \begin{cases} r_j & \text{if episode ends} \\ r_j + \gamma \max_{a'} Q'(s_{j+1}, a'; \theta') & \text{otherwise} \end{cases}$
  Perform gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
  if iteration mod $K = 0$ **then**
    $\theta' \leftarrow \theta$ ▷ Update target network
  end if
end for

weights that balance network latency against prediction quality, e_t is the current exit number, E is the total exits, $c_t \in \{-1, 1\}$ indicates prediction correctness, and L_t is the latency.

These weights are hyperparameters that we set through empirical evaluation but remained constant across all experiments.

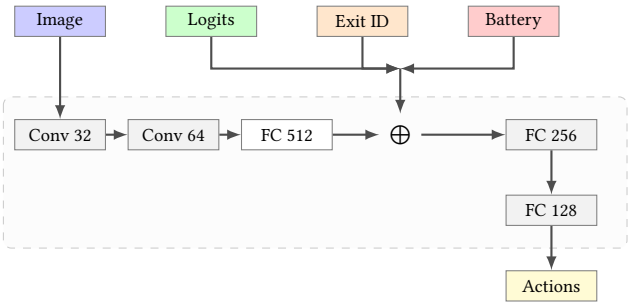


Figure 2: Architecture of the Deep Q-Network for early-exit decisions, processing the input image through convolutional layers and concatenating auxiliary information before selecting actions.

4 Simulation Environment

Our simulation framework utilizes SimPy[9], a process-based discrete-event simulation framework, to model the interactions between system components. The environment consists of three primary components:

- **Sensor Node**: Implements image capture and initial processing capabilities, including the early-exit neural network and decision-making agent.
- **Communication Channel**: modeled using a naive and simple Gaussian noise channel model [19]. The transmission delay follows:

$$t_{\text{delay}} \sim \mathcal{N}(\mu = 0.15\text{s}, \sigma = 0.02\text{s}) \tag{6}$$

- **Server**: Handles final classification when sensor nodes opt to offload computation.

Metric	Model	B0	B1	B2	B3
Time (s)	MobileNetV2	.008	.010	.010	.012
	ResNet18	.027	.023	.022	.041
MFLOPs	MobileNetV2	1.24	1.22	0.62	1.76
	ResNet18	154.1	134.9	134.6	134.4
mA	MobileNetV2	Proc: 487mA, Tx: 342mA			
	ResNet18	Proc: 690mA, Tx: 342mA			
KB	All	256	128	64	64

Idle: 324mA, Agent: 0.016s

Table 1: System Profile across different network models and their blocks on CIFAR-10

The sensor node components include a **battery model** tracking state-of-charge, voltage and temperature according to:

$$SoC_t = SoC_{t-1} - \frac{I \cdot \Delta t}{C_{\text{capacity}}} - R_{\text{discharge}} \cdot \Delta t \tag{7}$$

where I is current draw, Δt is time interval, $C_{capacity}$ is battery capacity, and $R_{discharge}$ is self-discharge rate. A camera module simulates image acquisition using cached datasets, while the processing unit models computational delays as:

$$t_{\text{processing}} = \sum_{i=1}^{n} t_{\text{block}_i} + t_{\text{agent}} \tag{8}$$

4.1 Performance Metrics

Our simulation framework tracks five critical performance metrics to evaluate system behavior. We measure classification **accuracy** against ground truth labels to assess model performance. **Capture-to-inference latency** accounts for both computational and transmission delays. The framework monitors energy consumption by profiling **current draw** across different operations. **Network usage** is quantified through the volume of data exchanged between sensor and server. Finally, we track **computational load** in FLOPS for each processing block to analyze resource utilization.

4.2 System Profiling

To ensure realistic simulation of resource constraints, we conducted extensive profiling using a Raspberry Pi 4 equipped with a PiJuice battery module[21] (1000mAh LiPo battery) as our reference sensor device platform. The profiling process characterized four key aspects of system behavior: power consumption through **current draw** measurements for processing and transmission operations, **processing times** via block-wise execution latency measurements, computational load by tracking **FLOP counts** for each network block, and **data transfer** based on feature map sizes at different network stages, see Table 1.

5 Experimental Setup

We evaluated our system using three standard image classification datasets with different architectures matched to their complexity. For **CIFAR-10**[7], a dataset of 60,000 32x32 color images across 10 classes, we employed ResNet18 [3] given its moderate computational requirements. We also employ Resnet18 for the more complex **Tiny ImageNet**[11], containing 200 classes and 100,000 64x64 images. Finally, we evaluated on **Visual Wake Words**[1] using MobileNetV2 [17], leveraging its efficiency and low footprint for this binary classification task of 115,000 images labeled as either "person" or "not-person". This dataset closely aligns with real-world IoT applications like presence detection.

We compare our Deep Q-Learning (DQL) approach against several baseline strategies to evaluate its effectiveness:

- **Always Transmit (AT)**: A naive baseline that performs minimal local computation and transmits data to the server for all inputs. This represents the traditional cloud-centric approach.
- **Early Exit (EE)**: A family of confidence-based thresholding strategies with varying aggressiveness setting each exit threshold to the same value. These strategies could be derived from application-specific requirements for accuracy and latency.

Strategy	Threshold
Early Exit Conservative	0.9
Early Exit Balanced	0.8
Early Exit Aggressive	0.6
Early Exit Very Aggressive	0.4

These baselines represent different points in the trade-off space between computational efficiency, accuracy, and resource utilization. The AT baseline provides a reference for traditional cloud offloading, while the EE variants demonstrate the impact of different static decision thresholds compared to our learned policy.

Strategy	Accuracy (%)	Latency (s)	Data Tx (MB)
MobileNetV2 On Visual Wake Words			
DQL	93.88	0.097	803.28
AT	91.78	0.201	1310.72
EC	91.24	0.058	0.0
EB	90.04	0.046	0.0
EA	84.64	0.031	0.0
EVA	82.16	0.027	0.0
ResNet-18 On CIFAR-10			
DQL	87.70	0.077	828.01
AT	85.44	0.199	1310.72
EC	84.64	0.057	0.0
EB	83.70	0.049	0.0
EA	78.78	0.036	0.0
EVA	72.40	0.029	0.0
ResNet-18 On Tiny ImageNet			
DQL	45.14	0.221	676255.54
AT	44.78	0.199	262144.00
EC	44.84	0.112	0.00
EB	44.74	0.111	0.00
EA	44.42	0.106	0.00
EVA	42.02	0.094	0.00

Table 2: Performance Comparison of Different Strategies on Benchmark Datasets

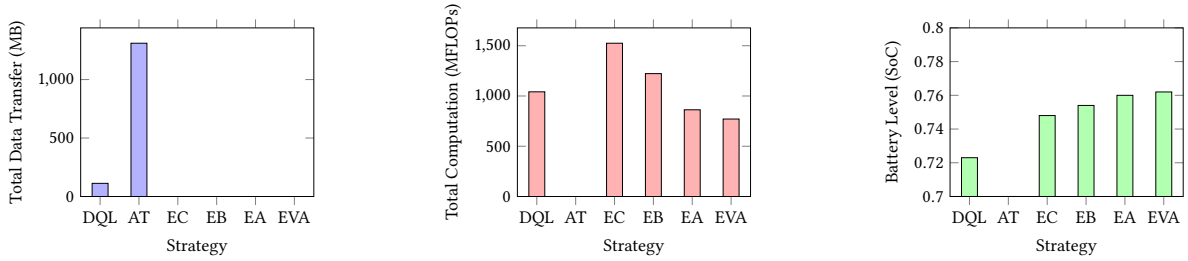


Figure 3: Resource usage comparison on MobileNetV2 with Visual Wake Words dataset

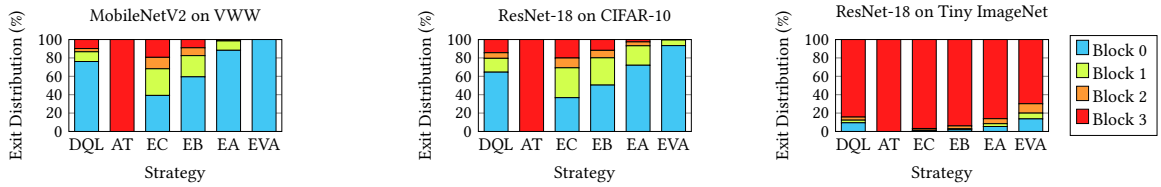


Figure 4: Exit distribution across strategies and datasets.

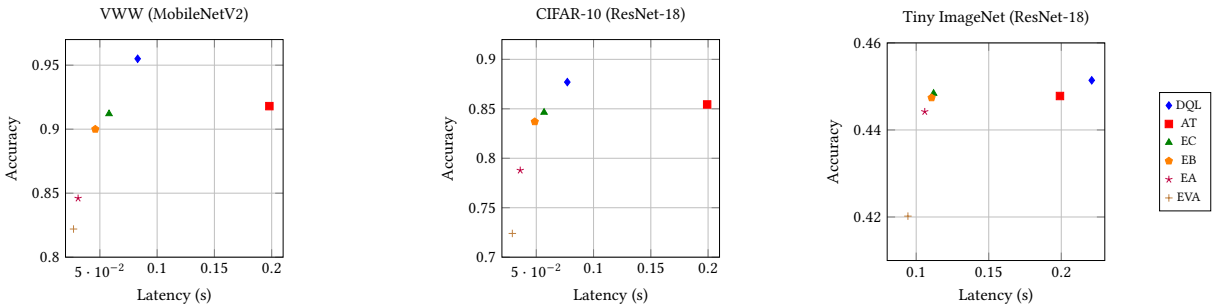


Figure 5: Accuracy vs. latency across datasets, models, and strategies.

6 Evaluation

We evaluated the performance of our ADAPT:DQL approach against baseline strategies across three benchmark datasets over 5000 simulation steps (equivalent to 1000 seconds of real-time at 1 FPS), as shown in Table 2.

Our results demonstrate that the DQL agent achieves superior accuracy compared to all baselines, including the Always Transmit strategy. This suggests that the agent successfully learns optimal exit points for specific images by leveraging both image features and logit vectors. The lightweight convolutional layers appear to function as an effective feature extractor, providing sufficient information alongside the logit vectors for the agent to make optimal exit decisions.

Figure 5 illustrates the trade-off between accuracy and latency in different strategies. As expected, more conservative exit thresholds yield higher accuracy at the cost of increased latency. For less complex data sets such as CIFAR-10 and Visual Wake Words, our DQL agent successfully replicates the traditional relationship between accuracy and latency, showing progressive improvements as latency increases.

The Tiny ImageNet dataset presents a unique challenge, with the full model achieving only 45% baseline accuracy. Due to the reward function's emphasis on accuracy, the agent struggles to learn a policy that disproportionately weights accuracy in the accuracy-latency trade-off.

The effect of various strategies on **resource usage** can be seen in Figure 3 in the case of MobileNetV2 on Visual Wake Words. The DQL agent performs a trade-off between local computation and data transmission seen by a modest increase in data transmission and battery usage and a decrease in computational load.

7 Discussion and Future Work

Our research demonstrates that a Deep Q-Learning agent can effectively learn optimal computation offloading policies, successfully balancing the trade-off between inference accuracy and latency. The empirical results reveal the agent's capability to adapt its exit strategies based on the inherent complexity of different datasets.

A comprehensive evaluation incorporating various exit placement configurations and network partitioning strategies would provide deeper insights into the framework's generalizability. For

example, depending on the dataset and model architecture, modifying exit placements may yield more representative logit vectors for the agent to make more optimal exit decisions. Such an analysis would elucidate the impact of architectural choices on the system's overall performance.

Future work should explore the integration of mobile sensor data and dynamic network conditions to enhance the agent's adaptability. This extension would better reflect real-world deployment scenarios and potentially lead to more resilient decision-making policies.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Nos. 1952247, 2133391, and 2420846.

References

[1] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. 2019. Visual Wake Words Dataset. arXiv:1906.05721 [cs.CV] https://arxiv.org/abs/1906.05721

[2] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv:1510.00149 [cs.CV] https://arxiv.org/abs/1510.00149

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[4] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2704–2713. doi:10.1109/CVPR.2018.00286

[5] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 615–629. doi:10.1145/3037697.3037698

[6] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. arXiv:1810.07052 [cs.LG] https://arxiv.org/abs/1810.07052

[7] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems* 25 (01 2012). doi:10.1145/3065386

[9] Stefan König, Tony Laird, Matthias Bors, et al. 2023. SimPy: Discrete Event Simulation for Python. https://simpy.readthedocs.io/ Version 4.0.1.

[10] Stefanos Laskaridis, Alexandros Kouris, and Nicholas D. Lane. 2021. Adaptive Inference through Early-Exit Networks: Design, Challenges and Directions. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning* (Virtual, WI, USA) *(EMDL'21)*. Association for Computing Machinery, New York, NY, USA, 1–6. doi:10.1145/3469116.3470012

[11] Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N* 7, 7 (2015), 3.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. doi:10.1109/5.726791

[13] Fang Liu, Guoming Tang, Youhuizi Li, Zhiping Cai, Xingzhou Zhang, and Tongqing Zhou. 2019. A Survey on Edge Computing Systems and Tools. *Proc. IEEE* 107, 8 (2019), 1537–1562. doi:10.1109/JPROC.2019.2920341

[14] Yoshitomo Matsubara, M. Levorato, and Francesco Restuccia. 2021. Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges. *Comput. Surveys* 55 (2021), 1 – 30. doi:10.1145/3527155

[15] Haseena Rahmath P, Vishal Srivastava, Kuldeep Chaurasia, Roberto G. Pacheco, and Rodrigo S. Couto. 2024. Early-Exit Deep Neural Network - A Comprehensive Survey. *ACM Comput. Surv.* (Oct. 2024). doi:10.1145/3698767 Just Accepted.

[16] Waseem Rawat and Zenghui Wang. 2017. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation* 29 (2017), 2352–2449. doi:10.1162/neco_a_00990

[17] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[18] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646. doi:10.1109/JIOT.2016.2579198

[19] Yuanming Shi, Kai Yang, Tao Jiang, Jun Zhang, and Khaled B. Letaief. 2020. Communication-Efficient Edge AI: Algorithms and Systems. *IEEE Communications Surveys and Tutorials* 22, 4 (2020), 2167–2191. doi:10.1109/COMST.2020.3007787

[20] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[21] Pi Supply. 2023. PiJuice: A Portable Project Platform For Every Raspberry Pi. https://github.com/PiSupply/PiJuice. Accessed: 2023-10-18.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[23] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. 2464–2469. doi:10.1109/ICPR.2016.7900006