Resource-Efficient Adaptive-Network Inference Framework with Knowledge Distillation-based Unified Learning

Rebati Gaire†, Sepehr Tabrizchi†, and Arman Roohi†
†School of Computing, University of Nebraska–Lincoln, Lincoln, NE, USA
{rgaire2, stabrizchi2, aroohi}@unl.edu

***Abstract*—Batteryless edge devices represent a promising avenue for sustainable computing, but are challenged by intermittent behavior and energy constraints. To address these issues, we propose a novel comprehensive approach integrating adaptive task module selection for intermittent computing paradigms. Our methodology incorporates the design of diverse task modules with varying sizes, precision levels, computational requirements, and energy consumption profiles, utilizing various compression techniques. These modules utilize a shared feature extractor that minimizes data movement and facilitates efficient checkpoint recovery, enhancing overall system robustness. In computing mode, the employed power-aware scheduler selects task modules based on performance requirements and available energy in the system. Subsequently, computation is performed to ensure optimal resource utilization while meeting application demands. We ensure optimal performance of these modules with proposed knowledge distillation-based unified learning. Quantitative evaluations on benchmark datasets—CIFAR-10, CIFAR-100, and Tiny-ImageNet—reveal that, with our proposed learning framework, designed models not only achieve improved performance metrics, including accuracy increases of 1.47%, 2.44%, and 3.70% for each dataset respectively but also enhance energy efficiency. These results validate our comprehensive and synergistic approach, demonstrating significant gains in both performance and resource optimization.**

***Index Terms*—Multi-task learning, model fine-tuning, intermittent computing, IoT**

I. Introduction

The rapid expansion of the Internet of Things (IoT) has led to an unprecedented growth in the number of connected devices, projected to reach 75 billion by 2025 [1]. These devices, equipped with diverse sensors, generate a massive volume of data at the edge of the network. The traditional approach of transmitting all this data to the cloud for processing is becoming increasingly unsustainable due to bandwidth limitations, latency concerns, and privacy issues. This has fueled the emergence of edge computing, a paradigm that pushes computation closer to the data sources, enabling real-time processing and decision-making [2]. The advent of deep learning has revolutionized the field of artificial intelligence, enabling machines to learn and make intelligent decisions from data. Deep neural networks (DNNs) have achieved remarkable success across various domains, including computer vision, natural language processing, and speech recognition. The integration of deep learning with edge computing, known as edge intelligence, holds immense potential for enabling smart and autonomous IoT applications. By deploying DNNs on edge devices, we can harness the power of local data to enable real-time insights, personalized experiences, and seamless interactions between the physical and digital worlds [3]. However, the deployment of DNNs on resource-constrained IoT devices presents significant challenges. DNNs are computationally intensive and require substantial memory and energy resources, which are often scarce on edge devices, while IoT devices are typically battery-powered. The high energy consumption of DNN inference can quickly drain the battery, limiting the operational lifetime of the devices. To address these challenges, various techniques have been proposed, such as model compression, hardware acceleration, and energy-efficient architectures. On the other hand, batteryless IoT devices, powered by energy harvesting from ambient sources such as solar, thermal, or RF, have emerged as a promising solution for sustainable edge computing. These devices scavenge energy from their environment, eliminating the need for battery replacement and maintenance. However, the intermittent and unpredictable nature of ambient energy sources poses unique challenges for computation. Batteryless devices are subject to frequent power failures, which can disrupt the execution of DNNs and lead to data loss. This intermittent computing paradigm requires novel approaches to ensure progress and maintain state across power cycles [4]. Several techniques have been proposed to enable reliable computation on intermittently-powered devices. Checkpoint-based approaches periodically save the system state to non-volatile memory (NVM), allowing resumption of execution after a power failure. However, these techniques often incur significant overhead and may not be suitable for the tight energy budgets of batteryless devices. Task-based approaches break down the computation into smaller, atomic tasks that can be executed within a single power cycle [5]–[7]. However, these techniques require careful task decomposition and scheduling, which can be challenging for complex DNN workloads.

This paper introduces an adaptive-network inference framework crafted to tackle the challenges of deploying neural networks in energy-constrained environments, notably batteryless IoT devices at the edge. Central to our approach is understanding that the energy consumption of neural networks scales with their size and computational intensity, as demonstrated in Figure 1. Our framework innovatively integrates adaptive task module selection with intermittent computing techniques. It features a range of task modules, each tailored to specific computational and energy profiles, enhanced by cutting-edge model compression techniques. A shared feature extractor across modules minimizes data transfers and supports efficient state recovery. Additionally, we integrate unified learning with

2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) | 979-8-3503-5411-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISVLSI61997.2024.00097

2159-3477/24/$31.00 ©2024 IEEE
DOI 10.1109/ISVLSI61997.2024.00097

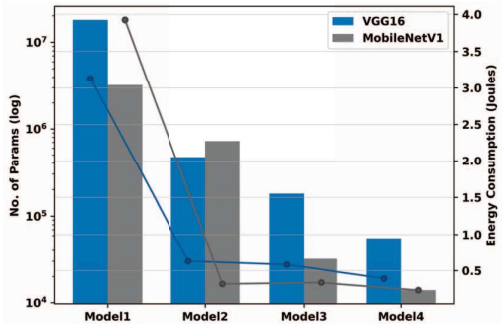

Fig. 1: Number of parameters vs. energy consumption for various task models with different number of parameters.

knowledge distillation to improve performance across multi-network systems, adjusting dynamically to the energy variability of batteryless devices. This method significantly enhances both the reliability and efficiency of resource-limited IoTs. The main contributions of this paper are: (1) An adaptive network inference framework with diverse task modules optimized for varied energy and performance requirements; (2) Exploration of training strategies for effective learning in a multi-task setting with a shared feature extractor; and (3) Comprehensive experimental evaluation demonstrating improved performance and resource efficiency on benchmark datasets.

II. Background

The rapid expansion of the IoT has led to an exponential increase in edge devices, generating massive streams of sensory data. Harnessing the insights from these data through ML techniques holds immense potential for applications across various domains. However, deploying computationally demanding DNNs on resource-constrained edge devices presents substantial challenges in terms of memory, energy, and latency. Network compression techniques [8]–[10] have emerged as a vital tool to address these constraints, enabling the creation of compact and efficient DNNs compatible with the limitations of edge hardware. Network compression encompasses a diverse range of strategies. Pruning methods aim to remove redundant connections (weights) or entire neurons from the DNN architecture, reducing model size. Quantization techniques decrease numerical precision (e.g., from 32-bit floating-point to 8-bit integers), leading to memory savings and faster computations. Knowledge distillation involves training a smaller "student" network to mimic the behavior of a larger, more accurate "teacher" network. Low-rank factorization methods approximate weight matrices using lower-rank decompositions, reducing computational complexity and storage requirements.

Energy-harvesting systems represent a critical advancement in the development of sustainable, autonomous computing devices, particularly within the Internet of Things (IoT). These systems derive power from environmental sources like solar radiation, thermal gradients, and ambient RF energy. The principle behind energy harvesting is to capture these omnipresent energies and convert them into electrical energy to power electronic devices. This approach enables devices to operate independently of conventional power grids, facilitating deployments in remote or inaccessible areas without regular maintenance. Energy-harvesting systems function intermittently, activating only when there is sufficient environmental energy and entering a state of power failure when the energy is insufficient. Therefore, the operation of energy-harvesting systems typically alternates between active periods and power-saving states. Devices are engineered to collect energy slowly, store it in elements like capacitors, and then consume this stored energy rapidly during active phases. This cycle presents unique challenges, especially the quick depletion of energy compared to its collection rate, which can lead to the loss of volatile memory states—such as registers and stack memory—during power outages, although NVM remains unaffected. The emergence of energy-harvesting neural network accelerators represents a significant innovation in the field of edge computing, particularly for devices that operate within the constraints of intermittent power sources. This advancement capitalizes on the local processing capabilities of CNNs, enabling edge devices to perform complex inference tasks autonomously. The move towards on-device computation is driven by the need to reduce latency, bandwidth usage, and reliance on constant cloud connectivity.

III. Proposed Approach

We introduce an adaptive-network inference framework with multiple task models, each with distinct computational complexity, energy profiles, and performance metrics, tailored to diverse energy availability and performance requirements. The structure of an intermittent-aware inference engine with the proposed adaptive-network inference framework is shown in Figure 2. A power-aware scheduler facilitates the selection of an appropriate network configuration for different scenarios by evaluating the available energy resources against the necessary performance metrics. Next, we outline the process of designing variable networks for specific tasks, detailing the strategic steps in crafting configurations optimized for diverse energy constraints and performance objectives. Subsequently, we discuss the methods used to train these networks effectively, ensuring optimal performance within defined energy parameters.

A. Network Design

To enhance reliability under intermittent energy conditions, we have designed neural networks featuring variable sizes, parameters, computational demands, and energy profiles specifically tailored to meet the requirements of distinct tasks. These networks leverage a power-aware scheduler that

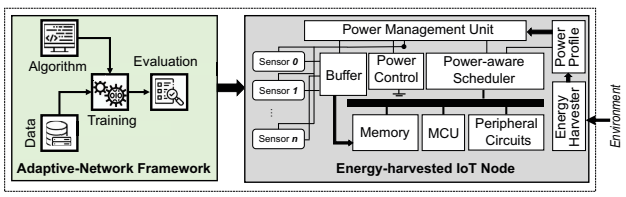

Fig. 2: Structure of an intermittent-aware inference engine.

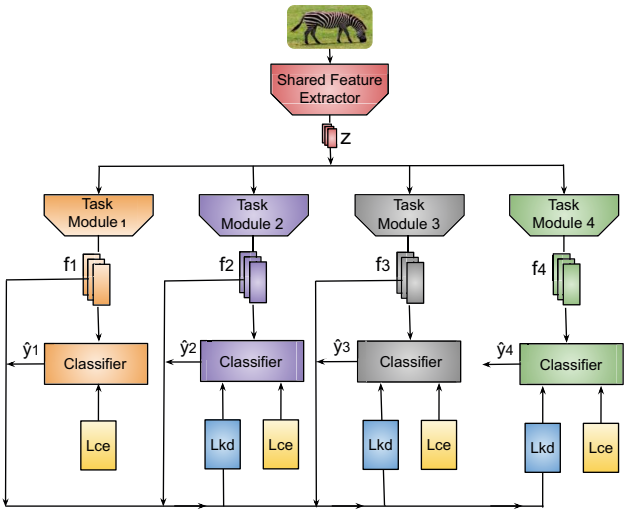

Fig. 3: The proposed unified learning framework. Each task module receives shared features z from a central feature extractor and processes them to generate specific features f_1, f_2, f_3, and f_4 along with predictions $\hat{y}_1, \hat{y}_2, \hat{y}_3$, and $\hat{y}_4$. These outputs are utilized to compute the knowledge distillation loss L_{kd} and the categorical cross-entropy loss L_{ce}, which collectively optimize the training of the entire framework.

dynamically selects the optimal modules for inference based on the prevailing energy availability and performance needs. We incorporate established deep neural network architectures, known for their robustness in edge inference applications, and utilize the initial convolutional blocks as a standardized feature extractor across all configurations. This feature extractor serves as the primary processing stage, efficiently extracting salient features from the input data that are rich in information and reduced in dimensionality relative to the original inputs. By utilizing a fixed feature extractor, we avoid redundant weight reloading when switching among different task modules, significantly reducing the energy consumed in data transfer. Moreover, such optimization facilitates energy-efficient feature checkpointing and enables smaller, task-specific modules to make precise predictions. The task modules are then subject to extensive model compression, achieving diverse operational characteristics through methods such as layer reduction, narrowing of layer widths, and the application of quantization and knowledge distillation, thus enhancing the networks' efficiency. Detailed specifications of each configuration—including parameters, memory demands, computations, and inference durations—are comprehensively documented in Section IV-B, supporting the deployment of these neural networks in energy-constrained environments.

B. Network Training

In this section, we outline various training strategies designed to optimize performance within an adaptive-network inference framework tailored for different energy scenarios. To ensure clarity and consistency throughout our discussion, we introduce several key terms and notations. The input sample is denoted by x, the true label by y, and the predicted label by $\hat{y}$. The parameters of any given module M are represented as Θ_M. For a task t, the function used for extracting features from the input is referred to as $g_t()$, while the task-specific post-processing function that converts these features into predictions is denoted by $f_t()$. We denote the categorical cross-entropy loss function for task t by $L_{ce_t}(y_t, \hat{y}_t)$, which is critical for training the network to minimize prediction errors. This function is expressed as:

$$L_{ce_t}(y_t, \hat{y}_t) = -\sum_{i=1}^{N_t}\sum_{c=1}^{C_t} y_{t,i,c}\log(\hat{y}_{t,i,c}) \quad (1)$$

where N_t and C_t are the number of samples and classes, respectively, for task t. The variable $y_{t,i,c}$ denotes whether the sample i for the task t actually belongs to class c and $\hat{y}_{t,i,c}$ denotes the predicted probability that the sample i for task t belongs to class c.

1) Baseline: Initially, we establish a baseline through independent training of each task model. Each task, indicated by k, is equipped with its dedicated feature extractor g_k and a task-specific module f_k. These components are trained separately, fostering the development of specialized feature representations and prediction mechanisms tailored for each task. The functional relationship for the output of each task, $\hat{y}_k$, given an input sample x, is defined by the following equation:

$$\hat{y}_k = f_k(g_k(x; \Theta_{FE_k}); \Theta_{TM_k}) \quad (2)$$

Here, the parameters Θ_{FE_k} and Θ_{TM_k} correspond to the feature extractor and the task module for the task k, respectively. The corresponding loss function for each task k is calculated as $L_{ce}(y_k, \hat{y}_k)$ from Equation 1. While this approach allows for high specialization through independent training, it results in each task having unique feature extractor parameters (Θ_{FE}), differing from one another. This specialization necessitates extensive data movement and energy consumption due to the frequent reloading of Θ_{FE} parameters each time a different task module is activated for inference. Consequently, this approach, although effective in isolation, poses significant challenges in terms of energy efficiency when deployed within a multi-network inference configurations.

2) Finetuning L2S: Subsequently, we explore approaches to maintain the Θ_{FE} stationery irrespective of the chosen task module. In this approach of fine-tuning from large to small (L2S), the training begins with the largest task module TM_L, utilizing a comprehensive feature extractor g. The output prediction for the largest task, $\hat{y}_L$ for an input sample x, is computed as follows:

$$\hat{y}_L = f_L(g(x; \Theta_{FE}); \Theta_{TM_L}) \quad (3)$$

Here, the parameters Θ_{FE} and Θ_{TM_L} correspond to the common feature extractor and the task module for the largest task L, respectively. The loss function, L_L, used for optimizing the parameters during training of TM_L, is defined as $L_{ce_L}(y_L, \hat{y}_L)$ in Equation 1. After training TM_L, the feature extractor's parameters, Θ_{FE}, are frozen. These pre-trained parameters are then employed to train each of the smaller

task modules. For each smaller task module k, the output prediction $\hat{y}_k$ for an sample x is calculated using the frozen feature extractor:

$$\hat{y}_k = f_k(g(x; \Theta_{FE_{frozen}}); \Theta_{TM_k}) \tag{4}$$

In this equation, Θ_{TM_k} are the parameters for the smaller task modules, and $\Theta_{FE_{frozen}}$ indicates that the feature extractor parameters remain unchanged. The loss function for the smaller tasks, L_k, is similar to that of the largest task and is given by $L_{ce_k}(y_k, \hat{y}_k)$ from Equation 1. This fine-tuning approach allows each task to benefit from a robust, generalized feature base, thus potentially increasing the overall efficiency and effectiveness of the multi-task learning framework.

3) Finetuning S2L: In contrast to the L2S strategy, the small to large (S2L) fine-tuning approach starts with the training of the smallest task module, TM_S, and focuses on utilizing a shared, fixed feature extractor across increasingly larger task modules. This method aims to explore how well small-scale learning can generalize to larger, more complex tasks. The initial phase involves training the TM_S, which is designed to manage the least complex scenarios. The output prediction for the smallest task, $\hat{y}_S$, from an input sample x, is derived as follows:

$$\hat{y}_S = f_S(g(x; \Theta_{FE}); \Theta_{TM_S}) \tag{5}$$

Here, the parameters Θ_{FE} and Θ_{TM_S} correspond to the common feature extractor and the task module for the smallest task S, respectively. The corresponding loss function, L_S, used to optimize the parameters during the training of TM_S, is defined by $L_{ce_S}(y_S, \hat{y}_S)$ in Equation 1. Once TM_S is adequately trained, the feature extractor's parameters, Θ_{FE}, are frozen. These pre-trained parameters are then used to independently train the larger task modules. This approach tests the ability of basic features, developed under limited complexity, to scale and adapt to more demanding scenarios. For each larger task module k, the prediction output $\hat{y}_k$ is calculated with the now frozen feature extractor:

$$\hat{y}_k = f_k(g(x; \Theta_{FE_{frozen}}); \Theta_{TM_k}) \tag{6}$$

Here, Θ_{TM_k} are the parameters for the larger task modules, and $\Theta_{FE_{frozen}}$ indicates that the feature extractor parameters remain unchanged. The loss function for the larger tasks mirrors that of the smallest task and is expressed as $L_{ce_k}(y_k, \hat{y}_k)$ in Equation 1. While this strategy provides certain benefits similar to L2S by exploiting a generalized feature base, it also faces limitations as the fundamental features derived from the smallest task may not capture the complexity required for optimal performance in larger tasks. Nevertheless, the capability of larger modules to accommodate and refine these initial features can sometimes result in improved performance over L2S, although it typically falls short of the baseline performance achieved with independently trained modules.

4) Unified Learning: In the unified learning approach, we diverge from traditional methodologies, where each task module is trained independently. Instead, all task modules are trained simultaneously using a shared feature extractor, denoted as g, which is trained concurrently across all tasks. This approach not only standardizes the weights across the feature extractor but also creates a synergistic learning environment where the learning outcomes from one task benefit others. The shared feature representation z extracted from an input x using the common feature extractor with parameters Θ_{FE} is given by: $z = g(x; \Theta_{FE})$. For each task k, the task-specific output $\hat{y}_k$ is then generated by the corresponding task module:

$$\hat{y}_k = f_k(z; \Theta_{TM_k}) \tag{7}$$

The overall loss function L, which optimizes the multi-task learning model, aggregates the losses from each task, weighted by their respective importance:

$$L = \sum_{k=1}^{K} \alpha_k L_{ce_k}(\hat{y}_k, y_k) \tag{8}$$

Here, α_k denotes the weight or importance assigned to the loss of each task k. These weights help balance the training focus among the tasks, depending on their significance and the complexity of the learning objectives. This multi-task framework leverages shared learning to minimize redundancy and maximize the efficiency of the model training process, demonstrating a significant advantage over isolated task-specific training models.

5) Unified Learning with Knowledge Distillation: Building on the unified learning framework, this enhanced version incorporates knowledge distillation to leverage the differential learning capabilities across networks of varying sizes. By using the more robust features and logits learned by larger networks, as depicted in Figure 3, this method optimizes the performance of smaller networks through guided learning from their larger peers. The approach maintains static weights for the common feature extractor and fosters an environment of mutual learning among the tasks, thereby elevating the overall performance beyond previous models.

For each task k, the task-specific output $\hat{y}_k$ continues to be generated by the respective task module as described in Equation 7. The comprehensive loss function L is now extended to integrate both the conventional task-specific losses and knowledge distillation losses:

$$L = \sum_{k=1}^{K} \left(\alpha_k L_{ce_k}(\hat{y}_k, y_k) + \sum_{j>k}^{K} \beta_{j,k} L_{kd_{j\to k}} \right) \tag{9}$$

Here, $\beta_{j,k}$ denotes the weight for each knowledge distillation loss. The knowledge distillation loss $L_{kd_{j\to k}}$, which facilitates the transfer of knowledge from a larger task j to a smaller task k, is defined by:

$$L_{kd_{j\to k}} = \|\phi_j(x) - \phi_k(x)\|_2 + \|\hat{y}_j - \hat{y}_k\|_2 \tag{10}$$

Here, $\phi_j(x)$ and $\phi_k(x)$ refer to the features from the teacher model and student model, respectively. This strategy not only enhances the efficiency of learning within smaller networks but also significantly boosts the overall system performance by harnessing the strengths of larger networks. By promoting the development of generalized features that are effectively applicable across various tasks, this learning model enhances both the efficiency and robustness of the learning outcomes.

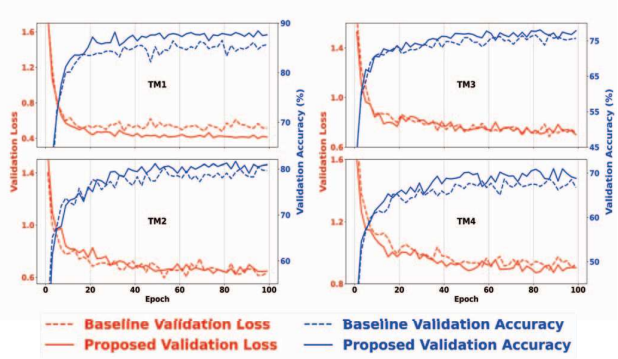

Fig. 4: Training convergence comparision of various task modules in Independent Learning (Baseline) and Proposed Learning Framework.

IV. Analysis and Evaluation

A. Dataset

Our methodology underwent rigorous evaluation using three prominent datasets in image recognition tasks: CIFAR-10 [11], CIFAR-100 [11], and Tiny-ImageNet [12]. CIFAR-10, renowned in computer vision research, includes 60,000 color images, each measuring $3\times32\times32$ pixels and categorized into ten distinct classes. CIFAR-100 broadens the spectrum by providing 60,000 color images of the same dimensions as CIFAR-10 but across 100 fine-grained classes, intensifying complexity and variability. In contrast, Tiny-ImageNet offers a downscaled version of the ImageNet dataset, comprising 200 object classes with 500 training samples per class. Resized to $3\times64\times64$ pixels, images in Tiny-ImageNet strike a balance between dataset intricacy and computational feasibility for computer vision experiments.

B. Baselines Architectures

Central to our innovative adaptive-network inference framework is implementing versatile network architectures underpinned by a shared feature extractor and an array of specialized task modules. This architecture draws upon the proven strengths of seminal deep neural network models, specifically VGG16 [13] and MobileNetV1 [14], acclaimed for their robust edge inference capabilities in image classification domains. Table I encapsulates the architectural essence and performance nuances of different designed task-specific modules. The task modules (TM) range from $TM1$, which mirrors the original full-scale models, embodying the peak of computational capability for accuracy-critical applications, to $TM4$, the most streamlined variant, which trims down to the bare essentials with few or no convolutional layers, optimizing for swift inference and minimal energy consumption in resource-constrained environments. Intermediate modules $TM2$ and $TM3$ mediate between these extremes, providing balanced options that cater to varying requirements for computational complexity and efficiency, all the while leveraging a shared feature extraction base (FE) to maintain consistency across the spectrum of network configurations.

C. Implementation Details

In our implementation, we employed the PyTorch framework for network creation, training, and testing, capitalizing on its versatility and efficiency. Optimization was achieved through stochastic gradient descent (SGD), a widely-utilized algorithm in deep learning. We initiated training with a learning rate of 0.01 for the task model, progressively reducing it by 90% after 80 epochs to enhance convergence. Training was conducted with a batch size of 128 over 200 epochs to ensure effective learning. Hyperparameters for the loss function were finetuned through grid search to optimize model performance. For our proposed joint training approach, we pre-trained the each network for twenty epochs using cross-entropy loss exclusively. This preliminary step facilitated the establishment of plausible intermediate features and logits before incorporating knowledge distillation losses. In determining the most effective instance of each task model, we preserved the best-performing checkpoint based on validation set performance. Subsequently, this checkpoint underwent rigorous evaluation on the test set to provide a comprehensive assessment of model generalization and performance.

D. Quantitative Evaluation

Figure 1 illustrates a comparison of various task models used in the image classification experiment, highlighting their size, measured by the number of parameters and corresponding energy consumption. To measure energy consumption, we employed the STM32F107VC microcontroller, providing accurate and reliable readings for our analysis. The graph clearly demonstrates a direct proportionality between model size and energy consumption. Table II presents a comprehensive performance comparison of various approaches for the adaptive-network inference framework described in Section III-B. From the table, we can clearly see that the employment of unified learning with a shared feature extractor and the inclusion of knowledge distillation led to a significant performance gain, often surpassing fine-tuning methods and the baseline.

Finetuning L2S resulted in decreased performance, particularly noticeable in streamlined modules with an average drop up to 34.58% compared to baseline performance on TinyImageNet classification with VGG16-based modules. This is attributed to the incompatibility of feature representations learned by more extensive task modules when applied to more diminutive counterparts. Conversely, Finetuning S2L showed improved results compared to the Finetuning L2S, with average accuracies dropping only 8.20% on the same task, suggesting that features learned by smaller networks are effectively expanded and refined by the larger ones due to their greater capacity. Unified training has demonstrated a clear beneficial impact on model performance across all datasets, with VGG16 showing average percentage improvements of 0.69% for CIFAR-10, 0.39% for CIFAR-100, and 0.87% for Tiny-ImageNet against the baseline, across all task modules. Similarly, MobileNetV1 exhibited gains of 0.73% for CIFAR-10, 1.46% for CIFAR-100, and 1.19% for Tiny-ImageNet, underlining the effectiveness of a holistic training

TABLE I: Detailed specification and performance evaluation of designed network modules.

DNN Model	Modules	Convolutional Layers			Fully-Connected Layers			Total		Inference	Module
		# Layers	# MAC	# Params	# Layers	# MAC	# Params	# MAC	# Params	Time (ms)	Size (MB)
VGG16	FE	2	158,072,832	38,720	0	0	0	158,072,832	38,720	8.136	0.156
	TM1	11	1,094,713,344	14,675,968	3	3,155,968	3,158,026	1,097,869,312	17,833,994	1.150	71.712
	TM2	2	94,371,840	369,024	2	66,176	66,250	94,438,016	435,274	0.319	1.202
	TM3	1	75,497,472	73,856	2	66,816	66954	75,564,288	140,810	0.227	0.611
	TM4	0	0	0	2	17,024	17,098	17,024	17,098	0.139	0.093
MobileNetV1	FE	1	884,736	928	0	0	0	884,736	928	7.842	0.005
	TM1	26	1,572,864,000	3,206,048	1	10,240	10,250	1,572,874,240	3,216,298	2.241	13.361
	TM2	10	126,877,696	713,248	1	10,240	10,250	126,887,936	723,498	1.296	3.302
	TM3	8	134,217,728	29,728	1	1,280	1,290	134,219,008	31,018	0.797	0.184
	TM4	4	92,274,688	11,680	1	1,280	1,290	92,275,968	12,970	0.107	0.539

TABLE II: Performance comparison of various designed network modules on different learning strategies. The best results are highlighted in bold, and the second-best results are shown in red color.

Network	Experiment	CIFAR-10				CIFAR-100				Tiny-ImageNet			
		TM1	TM2	TM3	TM4	TM1	TM2	TM3	TM4	TM1	TM2	TM3	TM4
VGG16	Baseline	89.38	85.09	81.92	75.74	63.57	**55.1**	53.36	45.31	54.20	48.47	38.37	31.95
	Finetuning L2S	89.38	81.32	75.34	53.51	63.57	51.00	47.98	30.18	54.20	26.6	21.54	16.18
	Finetuning S2L	90.19	84.37	80.59	75.74	65.84	51.62	49.49	45.31	55.19	37.69	33.61	31.95
	Unified Learning	90.59	85.07	82.76	76.04	65.44	53.82	53.15	45.94	56.07	47.66	38.04	32.77
	Unified Learning + KD (Ours)	**90.73**	**86.22**	**83.64**	**76.44**	**66.77**	54.76	**54.53**	**46.76**	**57.25**	**49.82**	**39.02**	**33.45**
MobileNetV1	Baseline	88.39	85.44	81.67	72.85	64.47	60.64	54.59	42.75	51.62	42.08	38.09	28.39
	Finetuning L2S	88.39	84.82	78.58	67.22	64.47	57.5	51.05	39.14	51.62	42.36	37.58	28.43
	Finetuning S2L	88.32	85.02	78.61	72.85	63.8	58.2	51.85	42.75	51.33	42.5	37.89	28.39
	Unified Learning	89.26	86.67	80.57	73.58	64.33	60.26	**55.09**	45.57	51.57	44.25	37.34	28.86
	Unified Learning + KD (Ours)	**90.30**	**87.65**	**82.48**	**75.23**	**64.77**	**61.29**	54.59	**46.32**	**51.88**	**46.23**	**38.60**	**29.49**

approach. The incorporation of knowledge distillation further amplified these improvements, resulting in more significant performance enhancements. With knowledge distillation embodied, VGG16's improvements surged to 1.47% for CIFAR-10, 2.44% for CIFAR-100, and 3.70% for Tiny-ImageNet. For MobileNetV1, the jumps were even more pronounced, with a 1.90% boost for CIFAR-10, 2.48% for CIFAR-100, and a remarkable 3.89% for Tiny-ImageNet. Furthermore, as evident in Figure 4, the convergence during training of the proposed approach with knowledge distillation demonstrates superior convergence during training, characterized by a better decline in validation loss and a higher rise in validation accuracy compared to the baseline. These experimental results underscore the synergistic advantage of our approach of unified training coupled with knowledge distillation, leading to more robust and generalizable models with greater training efficiency.

V. Conclusion

In this paper, we presented a comprehensive methodology for enhancing the efficiency and reliability of edge intelligence in batteryless IoT devices. Our approach, centered on adaptive task module selection and intermittent computing techniques, demonstrates significant advancements in managing the inherent constraints of power-scarce environments. Quantitative evaluations of the proposed learning framework using benchmark datasets like CIFAR-10, CIFAR-100, and Tiny-ImageNet revealed that our models not only achieve improved performance metrics but also enhance energy efficiency, with up to an average of 1.47%, 2.44%, and 3.70% of increase in accuracy for CIFAR-10, CIFAR-100, and Tiny-ImageNet classification, respectively, over the baseline. These results underscore our strategy's effectiveness in optimizing resource utilization while closely aligning with application-specific requirements.

Acknowledgement

This work is supported in part by the National Science Foundation under Grant No. 2303114 and 2247156.

References

[1] S. Al-Sarawi *et al.*, "Internet of things market analysis forecasts, 2020–2030," in *2020 Fourth World Conference on smart trends in systems, security and sustainability (WorldS4)*. IEEE, 2020, pp. 449–453.

[2] W. Shi *et al.*, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] Z. Zhou *et al.*, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[4] S. Umesh *et al.*, "A survey of techniques for intermittent computing," *Journal of Systems Architecture*, vol. 112, p. 101859, 2021.

[5] A. Roohi and R. F. DeMara, "Nv-clustering: Normally-off computing using non-volatile datapaths," *IEEE Transactions on Computers*, vol. 67, no. 7, pp. 949–959, 2018.

[6] N. Taheri *et al.*, "Intermittent-aware design exploration of systolic array using various non-volatile memory: A comparative study," *Micromachines*, vol. 15, no. 3, p. 343, 2024.

[7] S. Tabrizchi *et al.*, "Diac: Design exploration of intermittent-aware computing realizing batteryless systems," *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024.

[8] Y. Cheng *et al.*, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[9] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[10] R. Gaire *et al.*, "Encode: Enhancing compressed deep learning models through feature—distillation and informative sample selection," in *2023 International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2023, pp. 633–638.

[11] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," 2009.

[12] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[13] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[14] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.