2026 IEEE International Conference on Pervasive Computing and Communications (PerCom) | 979-8-3315-7613-4/26/$31.00 ©2026 IEEE | DOI: 10.1109/PerCom67906.2026.11524321

LEAF: Layer-Wise Energy-Adaptive Framework for Multi-DNN Real-Time Intermittent Systems

Hyunwoo Koo
Dept. of Computer Science and Engineering
Sungkyunkwan University
Republic of Korea
koowoo3@skku.edu

Wonyeong Lee
Dept. of Computer Science and Engineering
Sungkyunkwan University
Republic of Korea
lwy970327@skku.edu

Jinkyu Lee*
Dept. of Computer Science and Engineering
Yonsei University
Republic of Korea
jinkyu.lee@yonsei.ac.kr

***Abstract*—Intermittent systems powered by harvested energy face significant challenges due to limited memory and fluctuating energy availability. These challenges are further amplified in real-time applications such as environmental monitoring, where multiple DNN (Deep Neural Network) tasks must satisfy strict timing guarantees and maintain high inference accuracy. Existing solutions based on lightweight static or dynamic DNNs suffer from the lack of energy adaptability and/or excessive memory overhead, making them unsuitable for real-time intermittent systems. To address these limitations, we propose LEAF, a novel layer-wise energy-adaptive framework for real-time intermittent systems executing multiple DNN tasks. LEAF consists of three key components: (i) a runtime-reconfigurable DNN model and training methodology that supports multiple latency–accuracy trade-offs without incurring additional memory overhead; (ii) an energy-aware time budget abstraction that integrates intermittent inference behavior into existing timing guarantee frameworks; and (iii) a runtime scheduling mechanism that jointly leverages the model and abstraction to ensure both timing guarantees and high inference accuracy. Experimental results demonstrate that LEAF ensures the timely, energy-aware execution of multiple DNN tasks with higher accuracy compared to existing solutions, validating its effectiveness for real-time intermittent systems.**

I. Introduction

An intermittent system operates using energy harvested from the environment (such as RF, solar, or thermal energy), without relying on stable energy sources [1]–[3]. Their suitability for long-term remote or large-scale sensing deployments, where battery maintenance and waste are major burdens, has motivated the deployment of batteryless devices built on ultra-low-power MCUs (microcontroller units). As a result, these batteryless systems are increasingly adopted in applications like wildlife monitoring and bio-sensing, where multiple sensors perform on-device data analysis [4]. In such a context, consider a forest-deployed device tasked with periodic image analysis for environmental monitoring. Within each sensing interval, such a device typically executes multiple DNN (Deep Neural Network) tasks [5]. In these scenarios, DNN tasks must be completed on time to enable their timely inferences. To function reliably in these environments, real-time intermittent systems executing multiple DNN tasks must achieve the following key requirements:

R1. To guarantee the completion of multiple DNN tasks within their specified deadlines, and

R2. To maximize overall inference accuracy for multiple DNN tasks.

However, the deployment of multiple DNN tasks in real-time intermittent systems is fundamentally limited by the way of supplying energy and the resource scarcity of ultra-low-power MCUs, leading to the following critical constraints that make it challenging to satisfy R1 and R2.

C1. The nature of energy harvesting leads to fluctuations in both the amount and timing of available energy, and

C2. Intermittent systems are designed for ultra-low power, which severely limits the available memory space.

While no existing study fully addresses both R1 and R2 in the context of multi-DNN intermittent systems subject to C1 and C2, several prior studies are relevant to tackling these challenges. The first category of studies addresses either R1 [6], [7] or R2 [8] in isolation, but it fails to satisfy the other requirement and does not support multiple DNN tasks. The second category helps to address R1 under C1 and C2 for single-DNN inference by compressing and statically deploying lightweight DNN models [9], [10]. However, due to C1, even models using pruning techniques remain static and cannot adapt to fluctuating energy, thus failing to meet R2. The third category enhances adaptability to dynamic energy conditions by introducing runtime-adjustable architecture (such as model switching [11] or multi-exit mechanisms [12]). Nevertheless, these approaches cannot ensure timely DNN inference (thereby failing to meet R1) and incur additional memory overhead (thereby unfavorable to meet C2).

To overcome these limitations, this paper propose LEAF, a novel layer-wise energy-adaptive framework for real-time intermittent systems executing multiple DNN tasks. LEAF aims to address the following key issues:

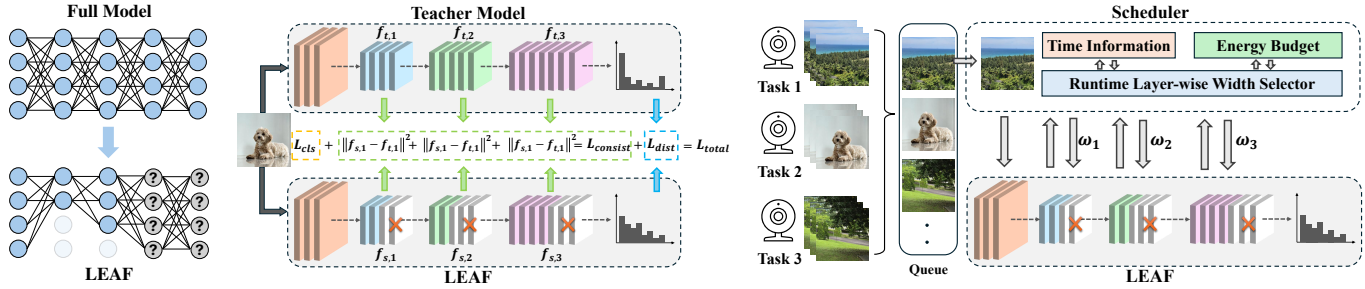
I1. How can we (re-)design a DNN model and its training process that (i) supports multiple latency-accuracy trade-off options, (ii) incurs no (or minimal) additional memory overhead, and (iii) enables dynamic runtime reconfiguration that leverages (i)?

I2. How can we characterize DNN inference behavior on intermittent systems and develop an interface that incorporates this behavior into the existing timing guarantee frameworks?

I3. How can we achieve both R1 and R2 under C1 and C2 by leveraging the solutions to I1 and I2?

*Jinkyu Lee (corresponding author) initiated this work while affiliated with Sungkyunkwan University.

979-8-3315-7613-4/26/$31.00 ©2026 IEEE



(a) Model architecture (b) Training phase (c) Run-time inference

Fig. 1: System overview of LEAF

For I1, we (re-)design a dynamic-width DNN architecture in Fig. 1(a) that allows for layer-wise adjustment of computational cost without requiring any additional weights, making it suitable for memory-constrained intermittent systems. In addition, we develop a consistency-aware training process in Fig. 1(b) to support runtime adaptation, enabling the model to dynamically adjust its configuration during inference based on energy and timing conditions. This tailored training procedure ensures reliable operation under runtime variability while maintaining high inference accuracy.

As to I2, we develop an energy-aware time budget abstraction that captures how intermittent power supply affects execution timing in a way compatible with existing real-time scheduling principles. By accounting for delays due to energy harvesting and intermittent execution overhead, this abstraction forms the basis for incorporating energy-related uncertainty into traditional real-time scheduling frameworks, which in turn enables systematic analysis and timing guarantees for DNN inference under fluctuating energy conditions.

For I3, we first ensure that the system can select feasible width configurations that meet task deadlines while preserving a guaranteed minimum level of inference accuracy. Beyond this baseline, we propose a runtime adaptation mechanism that dynamically selects the most accurate configuration feasible within the pre-established energy and timing budgets. Building upon the energy-aware time budget abstraction defined in I2, this mechanism explores candidate configurations at each decision point and chooses the one that maximizes inference accuracy without violating timely execution guarantees, as illustrated in Fig. 1(c).

Implementing LEAF on a TI MSP430FR5994 [13], experimental results demonstrate that LEAF achieves 100% deadline satisfaction even when executing 2–4 DNN tasks under dynamic energy conditions, and improves inference accuracy by up to 10% compared to static models with fixed widths. In contrast, the multi-exit model fails to operate beyond two DNN tasks due to memory overflow.

In summary, this paper makes the following contributions.

- We develop LEAF, the first attempt to address R1 and R2 in multi-DNN intermittent systems subject to C1 and C2.
- We adapt a memory-efficient DNN architecture and tailor its training process that enable dynamic layer-wise run-time adaptation to balance latency and accuracy (§III).
- We propose a novel abstraction that integrates energy harvesting behavior into existing real-time scheduling principles (§IV).
- We develop a scheduling framework that dynamically adjusts layer-wise widths at runtime while preserving offline real-time guarantees (§V).
- We demonstrate the effectiveness of LEAF through extensive real-world and simulation experiments (§VI).

II. Related Work

Static model compression for energy-efficient DNN inference. Recent studies have exploited pruning and quantization to lower DNN inference cost [9], [10]. These methods reduce memory footprint and inference latency to support DNN tasks on batteryless, intermittent devices under tight resource constraints. However, they focus on static memory/latency optimization and fail to adapt to dynamic changes in energy-harvesting conditions.

Dynamic neural networks for runtime adaptation. Recent research has introduced dynamic DNN models that leverage the trade-off between latency and accuracy by adjusting the network depth [12] or width [14], [15]. These methods are primarily designed for continuously powered systems. Depth-based approaches rely on coarse, pre-defined adaptation mechanisms, such as selecting among a limited set of exit points, while width-adaptive models typically select a fixed configuration prior to execution. As a result, existing dynamic DNNs provide limited fine-grained control of inference computation under intermittent execution. Approaches such as [16] apply depth-based early exits in energy-harvesting settings by selecting exit points based on energy and latency conditions. However, the additional model components incur memory overhead, limiting their applicability to memory-constrained, real-time intermittent systems.

Considering timeliness of tasks in intermittent systems. Several works [5], [6], [17] employ scheduling techniques aimed at maximizing the number of DNN jobs that meet their deadlines under energy constraints. Additionally, [8] uses reinforcement learning to improve deadline satisfaction. However, these approaches provide only best-effort timeliness and do not offer hard real-time guarantees.

In summary, existing studies lack both hard real-time guarantees and fine-grained adaptation. This leaves a gap for solutions that simultaneously provide granular flexibility and strict time-predictability.

III. Design of DNN Model and Training Process

In this section, we address I1 of the introduction. To this end, we re-design an existing dynamic-width DNN to make it suitable for runtime adaptation in intermittent systems. This involves modifying the model to satisfy the three qualifications explained in I1 of the introduction. We then develop the training process, which also supports the model's qualifications.

A. Qualified DNN Model Re-Design

As we explained in I1 of the introduction, we need a DNN model qualified as follows.

- *Latency-accuracy trade-off flexibility:* to support multiple options in DNN inference for exploring the latency-accuracy trade-off.
- *Memory efficiency:* to incur no (or as little as possible) additional memory overhead for DNN inference.
- *Runtime adaptivity:* to allow dynamic runtime reconfiguration for the latency-accuracy trade-off in DNN inference.

A common strategy for enabling latency-accuracy trade-offs in DNN inference is to adjust the model depth, as seen in multi-exit architectures [12]. These models attach auxiliary classifiers to intermediate layers, allowing early termination when prediction confidence is high. In our evaluation, we implement the multi-exit baseline according to the original design in [8], where exits are attached at intermediate layers. However, this design incurs non-trivial memory overhead because each exit adds extra parameters that scale with the feature dimensions of intermediate layers. As shown in Fig. 2 (labeled as ME), where the evaluation settings follow Section VI, our implementation adds two auxiliary exits, requiring 16 KB and 23 KB (in total 39 KB) of additional memory space, respectively. When added to the base model size of 31 KB (labeled as BASE), the total memory requirement increases to 70 KB, which is approximately 2.3× larger than the base model alone. Such memory requirements are difficult to accommodate on resource-constrained devices, making this approach less suitable for intermittent systems.

To overcome this limitation, we adopt a width-scalable design that adjusts inference cost by varying the width of each layer. This mechanism avoids structural changes and does not require additional weights. By selectively reducing the active width, the model enables fine-grained control over computation, lowering latency with minimal accuracy degradation. Although width-scalable models have been proposed before, they lack the ability to dynamically adjust width during inference. In our approach, we re-design the DNN model such that layer-wise width can be dynamically adjusted at runtime as shown in Fig. 1(a), allowing the model to scale its computational load based on system conditions. As a result, wider configurations yield higher accuracy at the cost of increased execution time. For example, as shown in Fig. 3, using the narrowest configuration (width 0.25 across all layers, meaning each layer computes at 25% of the full model capacity) achieves 62% accuracy with minimal latency, while using full width achieves 72% accuracy with 3.6× longer latency. Unlike

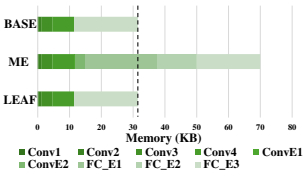


Fig. 2: Memory usage

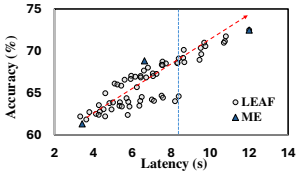


Fig. 3: L-A tradeoff

multi-exit architectures, which require additional parameters for each exit, our dynamic-width approach offers more fine-grained trade-off control while maintaining the same memory footprint as the base model as shown in Fig. 2 (labeled as LEAF), making it well-suited for deployment in memory-constrained intermittent systems.

While this dynamic-width architecture provides memory-efficient control over inference cost, the architecture itself cannot lead to runtime adaptivity. This introduces a key training challenge in ensuring that the model generalizes well across all dynamically selected width configurations. We address this challenge in the following subsection.

B. Training Process Development

We aim at developing a training process that supports the qualifications outlined above, with a particular focus on enabling runtime adaptation. Although prior studies such as US-Net [14] and OFA [18] have explored training methods for dynamic-width models, they assume a fixed width configuration during inference. As a result, these approaches cannot support fine-grained, runtime layer-wise reconfiguration needed in intermittent systems.

To address this limitation, LEAF aims to train a dynamic-width DNN that remains robust under varying runtime width configurations, designed to tackle two key challenges:

1) *Width-aware parameter adaptation:* to ensure that parameters can flexibly adapt to varying layer-wise width settings.
2) *Representation consistency:* to maintain stable internal feature representations, even when the active width selection changes dynamically.

LEAF extends the sandwich rule [14], which trains a network at its widest, narrowest, and a few random uniform widths across all layers. However, this uniform width strategy cannot expose the model to per-layer width combinations and produces inconsistent intermediate representations. LEAF solves these issues by sampling a different width for each layer in every iteration, adding a feature-level consistency loss ($\mathcal{L}_{\text{consist}}$ in Eq. (1)) to align sub-network features with a full-width teacher. To further support stable feature learning under reduced-width configurations, LEAF also adopts channel reordering [18], which rearranges channels within each stage based on their L1-norm importance, a widely adopted measure of importance, ensuring that narrow configuration preserve the most important channels.

These strategies are unified under a staged training, where the model is progressively exposed to wider ranges of width combinations while enforcing representational consistency. This design allows narrower sub-networks to learn from the representations of the full-width model, stabilizing optimization and preventing severe accuracy degradation that can arise

when all width configurations are trained jointly from scratch. The learning process is guided by a composite loss function (denoted by $\mathcal{L}_{\text{total}}$) that balances three objectives: prediction accuracy, knowledge transfer, and internal stability.

$$\mathcal{L}_{\text{total}} = \alpha \cdot \mathcal{L}_{\text{cls}} + (1-\alpha) \cdot \mathcal{L}_{\text{dist}} + \gamma \cdot \mathcal{L}_{\text{consist}}, \quad (1)$$

where α and γ are hyper-parameters that balance the three objectives. Each term in the objective serves a specific role:

- Classification loss ($\mathcal{L}_{\text{cls}}$): We used cross-entropy loss [19], defined as $\mathcal{L}_{\text{c}} = -\sum_{i=1}^{N_c} y_i \log \hat{y}_i$, where N_c is the number of classes, y_i is the true label (one-hot encoded), and $\hat{y}_i$ is the predicted probability. This loss encourages accurate predictions across all widths configuration.
- Knowledge distillation ($\mathcal{L}_{\text{dist}}$): We used soft-label distillation [20] to transfer the teacher's generalization to the student. The loss is defined as $\mathcal{L}_{\text{dist}} = -\sum_{i=1}^{N_c} p_{t,i} \log p_{s,i}$, where N_c is the number of classes, and $p_{t,i}$ and $p_{s,i}$ denote the softened probabilities of class i from the teacher and student networks, respectively. They are computed by applying temperature-scaled softmax: $p_i = \frac{\exp(z_i/\mathcal{T}_{temp})}{\sum_j \exp(z_j/\mathcal{T}_{temp})}$, where z_i (likewise z_j) is the logit for class i (likewise j), and $\mathcal{T}_{temp}$ is a temperature parameter that smooths the output distribution. This loss encourages the student to mimic the teacher's output distribution, maintaining accuracy across all width settings, even at reduced widths.
- Representational consistency ($\mathcal{L}_{\text{consist}}$): This loss is defined as $\mathcal{L}_{\text{consist}} = \sum_{l=1}^{N_L} \|f_{s,l} - f_{t,l}\|^2$, where $f_{s,l}$ and $f_{t,l}$ denote the intermediate feature maps at layer l from the student and teacher networks, respectively, and N_L is the total number of aligned layers [21]. It encourages consistent internal representations across widths, supporting stable inference even with narrow width configurations.

Our training in Algorithm 1 implements the strategies described above within a staged training process, where training proceeds over multiple stages. The network is initialized at full width to establish a high-accuracy baseline (Line 1). At each stage (s), a progressively wider width range $[w_{\min}^{(s)}, w_{\max}^{(s)}]$ is applied, allowing the model to gradually adapt to increasingly complex configurations (Line 2). LEAF is trained over a number of epochs (e) (Line 3), and each epoch iterates over batches (b) of training data (Line 4), where $\mathbf{X}_b$ denotes the input batch. For every batch, LEAF receives soft labels ($\hat{y}^{(t)}$) and intermediate feature maps ($F^{(t)}$) from the pre-trained teacher M_t (Line 5). LEAF is then exposed to a width range via the extended sandwich rule, generating Ω with minimum, maximum, and randomly sampled intermediate widths (Line 7). For each configuration (W_C) (Lines 8–11), LEAF computes the composite loss $\mathcal{L}_{\text{total}}$ by Eq. (1). The total loss is accumulated and backpropagated (Line 13). To support narrow-width configurations, we reorder channels at the end of each stage (except the last) based on L1-norm importance (Line 17).

Algorithm 1 Layer-wise width adaptation training

1: **Initialize**: LEAF at full width
2: **for** $s = 1 \rightarrow S$ **do** ▷ Stage s with widths $[w_{\min}^{(s)}, w_{\max}^{(s)}]$
3: **for** $e = 1 \rightarrow E$ **do**
4: **for** $b = 1 \rightarrow B$ **do**
5: $(\hat{y}^{(t)}, F^{(t)}) \leftarrow M_t(\mathbf{X}_b)$ ▷ Teacher outputs
6: $\mathcal{L} \leftarrow 0$
7: $\Omega \leftarrow \{w_{\min}^{(s)}, w_{\max}^{(s)}\} \cup \{\text{random}\}^{n_{\text{sandwich}}}$
8: **for** $W_C \in \Omega$ **do**
9: **Set width to** W_C
10: $(\hat{y}, F) \leftarrow \text{LEAF}(\mathbf{X}_b)$ ▷ LEAF outputs
11: $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_{\text{total}}$ in Eq. (1)
12: **end for**
13: **Update** parameters by backpropagating $\mathcal{L}$
14: **end for**
15: **end for**
16: **if** $s < S$ **then**
17: **Reorder channels** based on L1-norm importance
18: **end if**
19: **end for**

IV. Energy-Aware Time Budget Abstraction

This section develops an energy-aware time budget abstraction that is capable of integrating intermittent energy behavior into real-time scheduling principles. We first present our DNN inference task model, and then characterize the power-off and power-on cycles of intermittent systems. Based on this, we derive conservative time bounds that enable deadline guarantees for each DNN task under variable energy supply.

A. DNN Inference Task Model

In real-time intermittent systems, each DNN task $\tau_i \in \tau$ is periodically triggered with sensor input and specified as $(T_i, P_i^{\max})$, where T_i is the period (& the relative deadline), and $P_i^{\max}$ is the worst-case power consumption during execution. Each job (instance) of τ_i must be completed before the next job's release. Each task τ_i consists n_i layers, and the worst-case execution time (WCET) of each layer of τ_i varies depending on its selected width configuration. Let $C_{i,j}(w_{i,j})$ denote WCET of τ_i's j-th layer at width $w_{i,j}$, and $C_i(\mathbf{w}_i) = \sum_{j=1}^{n_i} C_{i,j}(w_{i,j})$ denote WCET of τ_i under a layer-wise width configuration $\mathbf{w}_i = (w_{i,1} \in W_{i,1}, \ldots, w_{i,n_i} \in W_{i,n_i})$, where $W_{i,j}$ is a set of candidate widths for τ_i's j-th layer.

B. Characterizing Intermittent Inference Behavior

In real-world harvesting environments, the energy supply is not deterministic but constantly varying. Consequently, for a task τ_i, the actual completion time under configuration $\mathbf{w}_i$ may exceed $C_i(\mathbf{w}_i)$ due to this continuous variability. To provide timing guarantees (R1) despite these fluctuations, we characterize the system behavior using conservative deterministic bounds derived from the worst-case energy conditions, which is illustrated by the execution pattern in Fig. 4.

Intermittently-powered devices remain off until the energy stored in the capacitor reaches a threshold E_{on}. During the power-off (harvesting only) phase $[t_0, t_1)$ as shown in Fig. 4, the system performs no computation and solely harvests energy. Once powered on, the device executes workload while simultaneously harvesting energy in the power-on (harvesting and execution) phase $[t_1, t_2)$. However, as the energy consumption during computation generally exceeds the harvesting rate, the net energy level decreases. When energy depletes, the device re-enters a power-off phase, repeating the cycle.

To maintain execution progress across power cycles, intermediate results in volatile memory (VM) must be explicitly saved to non-volatile memory (NVM) before shutdown.

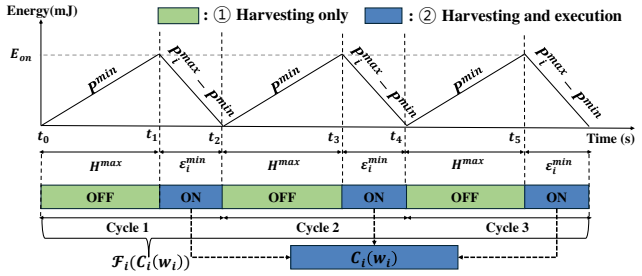


Fig. 4: Worst-case execution behavior in intermittent systems

This process, known as checkpointing, ensures correctness in subsequent reboots. We adopt static checkpointing at a fine-grained level, partitioning computations into atomic blocks and checkpointing only their outputs. If a shutdown occurs during the execution of an atomic block (mid-block), only the interrupted atomic block needs to be re-executed after reboot.

C. Developing Energy-Aware Time Budget Abstraction

We assume that the energy-harvesting system provides a minimum power input, as commonly adopted in prior intermittent computing studies studies [22]. Based on this assumption, intermittent systems alternate between power-off and power-on phases due to the nature of energy harvesting systems, as discussed in Section IV-B. During the power-off phase, computation halts entirely, which makes existing real-time scheduling principles that assume continuous execution, inapplicable. To enable real-time guarantees under these conditions, we develop energy-aware time budget abstraction that models energy harvesting delays explicitly.

Assuming a fixed-size capacitor, the amount of energy required to power on the system from V^{off} to V^{on} is calculated by $E^{\text{on}} = \frac{1}{2}C \cdot \left(V_{\text{on}}^2 - V_{\text{off}}^2\right)$, where C denotes the capacitance of the capacitor, V^{on} is the voltage threshold required to power on the system, and V^{off} is the voltage at which the systems shuts down. Then, the upper bound on the time required to harvest the energy amount to E^{on} is calculated by $H^{\max} = \frac{E^{\text{on}}}{P^{\min}}$, where $P^{\min}$ denotes worst-case energy input observed under realistic environmental conditions. As shown in Fig. 4, ① represents the power-off (harvesting only) phase during which the device remains off and harvests energy at a minimum rate of $P^{\min}$.

During execution, τ_i consumes energy stored in the capacitor while the system harvests additional energy. Therefore, the minimum guaranteed continuous execution of τ_i (denoted by $\mathcal{E}_i^{\min}$) is derived as follows. The power-on phase marked as ② in Fig. 4 represents the execution interval after power-on, during which energy decreases at a maximum rate of $P_i^{\max} - P^{\min}$ due to the net power consumption. This phase lasts for $\mathcal{E}_i^{\min}$, the minimum guaranteed on-time.

$$E^{\text{on}} + P^{\min} \cdot \mathcal{E}_i^{\min} = P_i^{\max} \cdot \mathcal{E}_i^{\min} \iff \mathcal{E}_i^{\min} = \frac{E^{\text{on}}}{P_i^{\max} - P^{\min}} \quad (2)$$

As illustrated in Fig. 4, each intermittent execution cycle consists of two distinct phases: ① a power-off (harvesting only) phase and ② a power-on (harvesting and execution) phase. These phases alternate until the task is completed. Since the energy available per cycle is limited, the task spans multiple cycles. The power-on phase includes overheads such as checkpointing and re-execution, which are necessary to ensure forward progress across power cycles. $O_i^{\max}$ represents an upper bound of the overhead for τ_i, which accounts for checkpointing, power-failure recovery, and the re-execution of interrupted atomic blocks under intermittent power.

Considering this cyclic phase structure, $\mathcal{F}_i(L)$ represents the maximum time to complete the L amount of execution of a job of τ_i, which includes all power-off and power-on phases, calculated as follows.

Theorem 1: Suppose that a job of τ_i is granted execution by the scheduler from t_0 to complete the L amount of execution, on an intermittent system. Then, the job of τ_i finishes the L amount of execution on an intermittent system, no later than $t_0 + \mathcal{F}_i(L)$, where

$$\mathcal{F}_i(L) = \left\lceil \frac{L}{\mathcal{E}_i^{\min} - O_i^{\max}} \right\rceil \cdot \left(H^{\max} + O_i^{\max}\right) + L, \quad (3)$$

Proof: In every power cycle, at most $H^{\max} + O_i^{\max}$ time is lost to power-off or overhead, while at least $\mathcal{E}_i^{\min} - O_i^{\max}$ time contributes to useful execution. Therefore, no more than $\left\lceil \frac{L}{\mathcal{E}_i^{\min} - O_i^{\max}} \right\rceil$ cycles are required to complete L amount of execution. Summing the maximum lost time over these cycles and adding L yields the stated bound. ■

Applying the theorem, $\mathcal{F}_i(C_i(\mathbf{w}_i))$ is an upper bound of time to complete a job of τ_i under a layer-wise width configuration $\mathbf{w}_i$. Note that all parameters for this abstraction can be obtained through offline profiling, to be demonstrated in Section VI-B.

V. Real-Time Scheduling Principles for Timing/Energy Guarantee & Accuracy Maximization

In this section, we present a scheduling framework for real-time intermittent systems that ensures deadline satisfaction (R1) and maximizes inference accuracy (R2). Based on the energy-aware time budget abstraction introduced in Section IV-C, we first develop how conservative timing guarantees for R1 can be established using static layer-wise configurations. We then extend this approach with a runtime adaptation mechanism that dynamically adjusts layer widths to improve accuracy while preserving deadline satisfaction.

A. Base Scheduling for Timing/Energy Guarantees

Theorem 1 can be applied to most (if not all) real-time scheduling principles, as long as they operate in a non-preemptive manner (due to the supposition of the theorem). In this subsection, we demonstrate how Theorem 1 can achieve R1 by applying it to an existing scheduling principle.

We target NP-FP (Non-Preemptive Fixed-Priority) scheduling and its schedulability analysis in [23]. Here, schedulability indicates whether all jobs meet their deadlines under the given scheduling policy. Under NP-FP, each task is assigned a fixed priority, which determines the order in which jobs start their executions; once a job starts its execution, no other job can preempt it. Let $\text{HP}(\tau_i)$ and $\text{LP}(\tau_i)$ denote the set of tasks with priorities higher and lower than τ_i, respectively.

To incorporate the behavior of intermittent systems into the NP-FP framework, we replace the original WCET $C_i(\mathbf{w}_i)$ with the intermittent-aware WCET $\mathcal{F}_i(C_i(\mathbf{w}_i))$, as explained in Theorem 1. This substitution allows us to directly reuse existing schedulability tests without changing the underlying analysis structure, as shown in the following lemma.

Lemma 1: Suppose that a DNN inference task set τ under a layer-wise width configuration $\{\mathbf{w}_k\}_{\tau_k \in \tau}$ on an intermittent system is scheduled by NP-FP. Then, any job of $\tau_i \in \tau$ does not miss its deadline, if the following is satisfied.

$$R_i \leq T_i, \tag{4}$$

where the worst-case response time R_i is calculated by a converged iteration such that $R_i^{(x+1)} = R_i^{(x)}$, starting from $R_i^{(0)} = \mathcal{F}_i(C_i(\mathbf{w}_i)) + \max_{\tau_j \in \mathrm{LP}(\tau_i)} \mathcal{F}_j(C_j(\mathbf{w}_j))$ using the following formula.

$$R_i^{(x+1)} = \mathcal{F}_i(C_i(\mathbf{w}_i)) + \max_{\tau_j \in \mathrm{LP}(\tau_i)} \mathcal{F}_j(C_j(\mathbf{w}_j)) + \sum_{\tau_h \in \mathrm{HP}(\tau_i)} \left\lceil \frac{R_i^{(x)}}{T_h} \right\rceil \cdot \mathcal{F}_h(C_h(\mathbf{w}_h)) \tag{5}$$

Proof: Theorem 1 proves that $\mathcal{F}_i(C_i(\mathbf{w}_i))$, $\mathcal{F}_j(C_j(\mathbf{w}_j))$ and $\mathcal{F}_h(C_h(\mathbf{w}_h))$ upper-bound the execution time of τ_i with $\mathbf{w}_i$, τ_j with $\mathbf{w}_j$ and τ_h with $\mathbf{w}_h$, respectively on an intermittent system. Lemma 1 in [23], which is an offline schedulability test for NP-FP, is equivalent to this lemma by replacing the original WCET of τ_i, τ_j and τ_h to $\mathcal{F}_i(C_i(\mathbf{w}_i))$, $\mathcal{F}_j(C_j(\mathbf{w}_j))$ and $\mathcal{F}_h(C_h(\mathbf{w}_h))$, respectively. Therefore, this lemma holds by Lemma 1 in [23] and Theorem 1. ∎

Note that Lemma 1 in [23] (and therefore Lemma 1 in this paper) uses a popular response time analysis framework, in which the first, second and third terms in the right-hand-side of Eq. (5) respectively present the WCET of τ_i itself, the WCET of a lower-priority blocking job, and the sum of the WCETs of higher-priority jobs.

B. Runtime Adaptation for Accuracy Maximization

While Lemma 1 guarantees R1 for a given layer-wise width configuration $\{\mathbf{w}_i\}_{\tau_i \in \tau}$, it fails to achieve R2. Achieving R2 is particularly challenging in intermittent systems, where energy availability varies due to alternating between power-off/on phases, limiting the duration of continuous operation as discussed in Section IV-B. In this context, simply increasing the model width to improve accuracy can easily lead to deadline misses due to insufficient execution time within power cycles. This calls for a runtime mechanism that expands widths beyond the given configuration only when energy-aware time budget allows, while still preserving timely execution for all tasks to be released in the future.

There exists a runtime mechanism based on NP-FP scheduling [23], but it is not suitable for intermittent systems. In this approach, once a job τ_i is selected by the scheduler, the available time budget is computed only once at the start and remains fixed throughout execution. However, intermittent systems experience time budget changes across layers due to fluctuations in harvested energy. Our DNN model's layer-by-layer design allows execution to adapt at each boundary, requiring finer-granularity runtime decisions to match the current energy availability without violating timing constraints.

Algorithm 2 Runtime width re-selection for remaining layers

1: **Input:** Task τ_i to be executed, current time t_0, deadline t_D, remaining candidate width set $\mathcal{W}_i^{\mathrm{rem}}$, accuracy function $\mathrm{Acc}(\cdot)$
2: **Output:** Selected width configuration $\mathbf{w}_i^{\mathrm{rem}} = (w_{i,r}^{\mathrm{rem}}, \ldots, w_{i,n_i}^{\mathrm{rem}})$
3: Initialize $\mathbf{w}_i^{\mathrm{rem}}$, $\Delta_i \leftarrow t_D - t_0$ ▷ Time budget for τ_i
4: **for all** $\mathbf{w}_i' = (w_{i,r}', \ldots, w_{i,n_i}') \in \mathcal{W}_i^{\mathrm{rem}}$ **do**
5: $\quad C_{i,r}^{\mathrm{rem}}(\mathbf{w}_i') \leftarrow \mathcal{F}_i\big(\sum_{u=r}^{n_i} C_{i,u}(w_{i,u}')\big)$
6: $\quad$ **if** $C_{i,r}^{\mathrm{rem}}(\mathbf{w}_i') \leq \Delta_i$ **and** Lemma 2 holds of all $\tau_j \in \tau \setminus \{\tau_i\}$ **then**
7: $\quad\quad$ **if** $\mathrm{Acc}(\mathbf{w}_i') > \mathrm{Acc}(\mathbf{w}^{\mathrm{rem}})$ **then**
8: $\quad\quad\quad$ Update $\mathbf{w}_i^{\mathrm{rem}} \leftarrow \mathbf{w}_i'$
9: $\quad\quad$ **end if**
10: $\quad$ **end if**
11: **end for**

1) After a job of τ_i has been selected for execution according to NP-FP, the system must determine, at the beginning of each layer, whether and how to expand the width beyond the remaining configurations in $\mathbf{w}_i$. This introduces multiple decision points per job, corresponding to the number of layers in the model, without compromising scheduling principles for NP-FP.
2) At each decision point, the system must carefully allocate the remaining time budget across the unexecuted layers in $\mathbf{w}_i$. The goal is maximizing inference accuracy without violating the job's deadline. This is a non-trivial problem, as the optimal choice depends not only on the remaining time budget but also on the expected contribution of each layer's width to the final accuracy.

The importance of this decision is supported by experimental evidence. As shown by the dashed blue vertical line in Fig. 3, multiple width configurations yield similar latency (around 8.3 seconds), but vary greatly in accuracy (64.0% to 69.1%), depending on per-layer width allocation. This demonstrates that accuracy depends not only on total latency but also on width distribution. To enable such fine-grained adaptation, we propose a runtime width re-selection algorithm that dynamically determines the best configuration for the remaining layers of each job. Algorithm 2 describes the procedure for selecting optimal width configurations for the remaining layers of task τ_i at runtime.

This runtime width re-selection mechanism is invoked at the beginning of each layer execution, but only applies to the layers of τ_i that have not yet been executed. The remaining configuration of τ_i is dynamically determined to maximize inference accuracy while satisfying the task deadline. The remaining time budget Δ_i as the difference between the task deadline t_D and the current time t_0. The algorithm begins by initializing the remaining width configuration $\mathbf{w}_i^{\mathrm{rem}}$ to a narrowest setting (Line 3). Let $\mathcal{W}_i^{\mathrm{rem}}$ denote the set of all possible width configurations for the remaining layers of task τ_i, formally defined as $\mathcal{W}_i^{\mathrm{rem}} = \left\{(w_{i,r}, \ldots, w_{i,n_i}) \;\middle|\; w_{i,j} \in W_{i,j} \text{ for } j = r, \ldots, n_i\right\}$ where $W_{i,j}$ is the set of candidate widths for layer j (Line 4). It then iterates over all possible width combinations $\mathbf{w}_i' \in \mathcal{W}_i^{\mathrm{rem}}$, where each $\mathbf{w}_i'$ represents a candidate width vector for the remaining layers. For a given candidate configuration

$\mathbf{w}'_i = (w'_{i,r}, \ldots, w'_{i,n_i})$, we define the total execution cost as $C^{\text{rem}}_{i,r}(\mathbf{w}'_i) = \mathcal{F}_i\big(\sum_{u=r}^{n_i} C_{i,u}(w'_{i,u})\big)$ (Line 5).

The candidate is considered schedulable if it satisfies two conditions: (i) its execution time does not exceed the remaining time budget Δ_i, and (ii) it does not cause any other jobs to miss their deadlines by a run-time schedulability test (Line 6). We adopt a run-time schedulability test for NP-FP [23], and apply it to our setting by substituting the WCET term with our width-dependent intermittent-aware WCET by applying $\mathcal{F}_i(\cdot)$. With this substitution, the corresponding schedulability test in our setting is given by Lemma 2. Among the schedulable candidates for τ_i, the algorithm selects the configuration that yields the highest expected accuracy, based on a pre-computed accuracy table obtained from offline profiling (Lines 7–8).

To check the condition (ii), we extend Lemmas 3 and 4 in [23], to a finer-grained layer level, developing Lemma 2 that checks at each layer whether increasing τ_i's width preserves the schedulability of the other task τ_j. For Lemma 2, we need the following concepts and notations. A job is said to be active at time t_0 if it has been released no later than t_0 and has remaining execution at t_0. Let $A_j(t_0) \in \{\mathtt{T}, \mathtt{F}\}$ indicate whether task τ_j has an active job at time t_0, and let $r_j(t_0)$ denote its earliest release time of τ_j at or after t_0, which also serves as the absolute deadline of τ_j if $A_j(t_0) = \mathtt{T}$.

Lemma 2: Suppose we apply the run-time adaptation mechanism in Algorithm 2, to a task set τ under a layer-wise width configuration $\{\mathbf{w}_k\}_{\tau_k \in \tau}$, which satisfies Lemma 1. The earliest active job after t_0 of task $\tau_j \notin \tau \setminus \{\tau_i\}$ under a layer-wise width configuration $\mathbf{w}_j$ does not miss its deadline, if the following holds when τ_i's r-th layer starts its execution at t_0.
(Case 1) when $A_j(t_0) = \mathtt{T}$,

$$F_j(C_j(\mathbf{w}_j)) + C^{\text{rem}}_{i,r}(\mathbf{w}^{\text{rem}}_i) + \sum_{\substack{\tau_h \in \text{HP}(\tau_j)\setminus\{\tau_i\} \\ |A_h(t_0)=\mathtt{T}}} F_h(C_h(\mathbf{w}_h)) + \sum_{\substack{\tau_h \in \text{HP}(\tau_j) \\ |r_h(t_0)<r_j(t_0)}} \left\lceil \frac{r_j(t_0) - r_h(t_0)}{T_h} \right\rceil \cdot F_h(C_h(\mathbf{w}_h)) \le r_j(t_0) - t_0 \quad (6)$$

(Case 2) when $A_j(t_0) = \mathtt{F}$,

$$F_j(C_j(\mathbf{w}_j)) + C^{\text{rem}}_{i,r}(\mathbf{w}^{\text{rem}}_i) + \sum_{\substack{\tau_h \in \text{HP}(\tau_j)\setminus\{\tau_i\} \\ |A_h(t_0)=\mathtt{T}}} F_h(C_h(\mathbf{w}_h)) + \sum_{\substack{\tau_h \in \text{HP}(\tau_j) \\ |r_h(t_0)<r_j(t_0)+T_j}} \left\lceil \frac{r_j(t_0) + T_j - r_h(t_0)}{T_h} \right\rceil \cdot F_h(C_h(\mathbf{w}_h)) \le r_j(t_0) + T_j - t_0 \quad (7)$$

Proof: Theorem 1 proves that $\mathcal{F}_j(C_j(\mathbf{w}_j))$ and $\mathcal{F}_h(C_h(\mathbf{w}_h))$ upper-bound the execution time of τ_j with $\mathbf{w}_j$ and τ_h with $\mathbf{w}_h$, respectively on an intermittent system. According to Line 5 of Algorithm 2, $C^{\text{rem}}_{i,r}(\mathbf{w}^{\text{rem}}_i) = \mathcal{F}_i\big(\sum_{u=r}^{n_i} C_{i,u}(w^{\text{rem}}_{i,u})\big)$ calculates the WCET of layers r to n_i of τ_i under a layer-wise width configuration $\mathbf{w}^{\text{rem}}_i$. Lemmas 3 and 4 in [23], which is a run-time schedulability test, are equivalent to this lemma by replacing the WCET to applying the $\mathcal{F}(\cdot)$ function in Eq. (3). Thus, this lemma holds by Lemmas 3 and 4 in [23], Theorem 1, and the calculation of $C^{\text{rem}}_{i,r}(\mathbf{w}^{\text{rem}}_i)$. ■

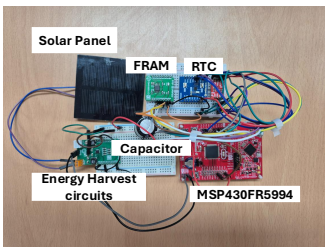


Fig. 5: Hardware setup

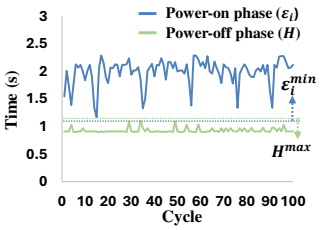


Fig. 6: Actual energy pattern

While Lemma 2 applies only to the earliest active job after t_0 of each task $\tau_j \notin \tau \setminus \{\tau_i\}$, it is shown in [23] that all subsequent jobs of tasks $\tau_k \in \tau$ (including τ_i itself) also remain schedulable if all subsequent jobs of $\tau_k \in \tau$ execute with the width configuration $\mathbf{w}_k$ in the supposition of Lemma 2 (while the only exception is the job of τ_i to be executed at t_0 with $\mathbf{w}^{\text{rem}}_i$).

Run-time complexity. Algorithm 2 checks schedulability for each remaining-layer width combination. This involves evaluating Eqs. (6) or (7), each of which incurs a computational cost of $O(n^2)$ per iteration, where n is the number of DNN tasks. The number of width combinations M is determined by the product of the number of candidate widths for each remaining layer, formally given as $M = \prod_{u=r}^{n_i} |W_{i,u}|$. Therefore, the overall time complexity is $O(M \cdot n^2)$. In our experiments, the number of configurable layers is limited to three, and each layer has four width candidates, resulting in only M=4^3=64 total combinations. Since each width combination only incurs $O(n^2)$ computations, the total computational overhead remains very low. In practice, the entire re-selection procedure completes within a few milliseconds (to be detailed in Section VI-B), which makes it practical even in real-time intermittent systems.

VI. Evaluation

This section presents the implementation of LEAF and compares its performance with existing approaches under both real-world and simulated settings.

A. Implementation

Hardware setup (shown in Fig. 5). LEAF runs on an MSP430FR5994 evaluation board (8 KB SRAM, 256 KB FRAM, LEA accelerator) at up to 16 MHz, using its onboard ADC to monitor voltage. An AM1805 RTC shares the system capacitor to maintain time across power failures. Input data is stored and retrieved from an external FRAM module. We emulate solar energy harvesting using a 300W halogen lamp and a TI BQ25570 module connected to 4.7mF capacitors. The board is powered at 2.3V and turning on when the capacitor voltage exceeds 3.25V and shutting down below 2.85V.

Model development. We design a LeNet-inspired DNN [24] with four convolution layers and one fully connected layer to enable fine-grained control via layer-wise width adjustment, as summarized in Table I. The first layer width is fixed to ensure minimal feature extraction, and the width of the final

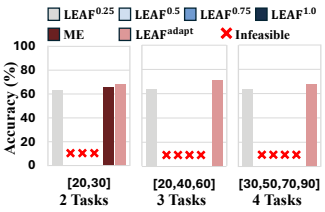


Fig. 7: Case study results

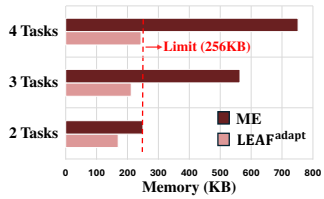


Fig. 8: Memory usage

fully-connected layer is set to match that of the preceding layer to fully utilize the extracted features. The model implemented in PyTorch (v2.4.0) and int8-quantized via ONNX, reducing size from 132 KB to 31 KB. The multi-exit model adds early exits after the first and second convolution layers, increasing size to 70 KB. Models are trained and evaluated on CIFAR-10 [25]. We reserve 10% of the training data for validating width configurations and use 90% for training. We set parameters in Eq. (1) as $\alpha = 0.5$, $\gamma = 0.1$ and $\mathcal{T}_{temp} = 2$.

Method implementation. We use FreeRTOS [26] as the underlying real-time operating system to manage tasks and timing services. On top of this, we implement a custom scheduler that enforces non-preemptive, fixed-priority (NP-FP) scheduling, using Rate Monotonic (RM) as the priority assignment policy. Each task is assigned a static priority based on its period, and the scheduler handles task release, deadline checking, and layer-wise execution control accordingly. Accuracy for each width configuration is obtained from validation set results and used for runtime selection.

B. Case Study

Real-world setup. We evaluate our system on the MSP430FR5994 under realistic intermittent energy conditions using a solar energy harvester. We additionally verified the system's reliability under practical conditions by using an artificial light source to emulate real-world environments. As shown in Fig. 7, experiments are conducted with two to four multi-DNN tasks, with periods [20 and 30], [20, 40 and 60], and [30, 50, 70 and 90] in seconds. We execute 200 jobs per configuration and measure the ratio of correctly completed jobs, which meet their deadlines and produce a correct inference result (the predicted label matches the ground truth for classification). Results with deadline misses or memory overflows are marked as "x" in the figure (called *infeasible*).

TABLE I: DNN architecture (layer-wise width-scalable)

Layer	Type	Output Channels	Output Size
Input	–	3	32×32
Conv1	Conv2D + ReLU	6	32×32
Conv2	Conv2D + ReLU	≤16	16×16
Conv3	Conv2D + ReLU	≤24	8×8
Conv4	Conv2D + ReLU	≤32	8×8
FC	Fully Connected	10	1×1

TABLE II: WCETs and accuracy for width configurations

$(w_1, w_2, w_3, w_4, w_5)$	WCET (s)	Accuracy (%)
(1.00, 0.25, 0.25, 0.25, 0.25)	3.37	62.2
(1.00, 0.25, 1.00, 1.00, 1.00)	8.45	64.6
(1.00, 0.50, 0.50, 0.50, 0.50)	5.31	66.0
(1.00, 0.75, 0.75, 0.75, 0.75)	8.71	70.1
(1.00, 1.00, 1.00, 1.00, 1.00)	12.04	72.4

Offline profiling. To obtain execution time for each DNN task, we perform offline profiling on the MSP430FR5994 under continuous energy supply, where power is maintained in a stable state without interruptions or power cycles. This means, the profiled inference time excludes harvesting delays; under intermittent power supply, the completion time for inference is much longer as we explained in Section IV-B. Regarding the profiled inference time, Layer 1 takes 1.96 s, while the other layers vary: layer 2 from 0.65 s to 2.50 s, layer 3 ranges from 0.27 s to 2.25 s, layer 4 from 0.45 s to 4.89 s, and layer 5 (fc layer) from 0.04 s to 0.17 s. We profile the scheduling overhead, which remains low: under 10 ms for up to three tasks and below 20 ms even with four concurrent tasks.

Table II presents WCET and accuracy for representative width configurations. In general, increasing the total width across layers tends to result in improved inference accuracy, which however does not always hold. Even with increased execution time, a configuration may yield lower accuracy; for instance, while $(w_1, w_2, w_3, w_4, w_5) = (1.0, 0.25, 1.0, 1.0, 1.0)$ takes 8.45 s and yields 64.60% accuracy, $(1.0, 0.50, 0.50, 0.50, 0.50)$ requires only 5.31 s and achieves 66.00% accuracy.

We obtain all time and energy parameters through offline profiling under realistic intermittent conditions, including power-cycle overhead and scheduler latency. The energy required to power on the device is $E_{\text{on}} = 4.97$ mJ, and task-level re-execution incurs a worst-case overhead of $O_i^{\max} = 80$ ms, which also includes scheduler latency. The minimum and maximum power rates are $P^{\min} = 4.56$ mW and $P_i^{\max} = 9.71$ mW. To parameterize the energy-aware time budget model, we conservatively estimate two key bounds over 100 power cycles using a real-time clock (RTC): the minimum uninterrupted execution duration $\mathcal{E}_i^{\min} = 0.97$ s, and the maximum harvesting time $H^{\max} = 1.09$ s, which represents the longest time needed to accumulate sufficient energy for a power-on cycle. As shown in Fig. 6, these bounds define each cycle's execution and harvesting phases, forming the basis for the case study results in Fig. 7 (as well as the simulation results for Section VI-C).

Approaches to be compared. We evaluate six system configurations, all sharing the same scheduling policy (i.e., NP-FP) and offline schedulability analysis (i.e., Lemma 1), four of which do not have any run-time adaptation mechanism.

- `LEAF`$^{0.25}$, `LEAF`$^{0.5}$, `LEAF`$^{0.75}$ and `LEAF`$^{1.0}$: static models without any run-time mechanism with fixed maximum widths of 0.25, 0.5, 0.75 and 1.0, respectively.

Two approaches apply our runtime adaptation mechanism.

- `LEAF`$^{\text{adapt}}$: our proposed model that adjusts the width of each layer by the runtime mechanism in Algorithm 2 and serves as the practical implementation of LEAF.
- `ME`: Multi-exit DNN model that supports multiple internal exits for early termination [12] by the runtime mechanism corresponding to Algorithm 2.

Real-world results. As shown in Fig. 7, `LEAF`$^{\text{adapt}}$ always yields the highest accuracy for every case. In particular, when executing two tasks, `LEAF`$^{0.25}$ achieved 63.5% accuracy, while `LEAF`$^{\text{adapt}}$ achieved 68.5%. `ME` attained 66%

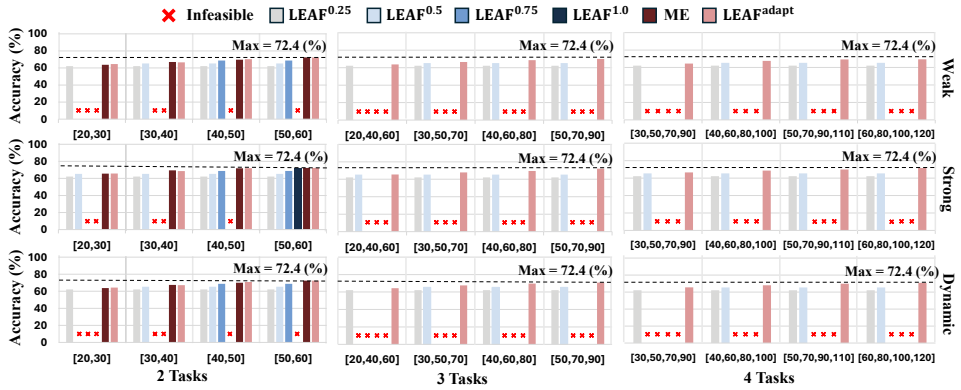


Fig. 9: Simulation-based evaluation results

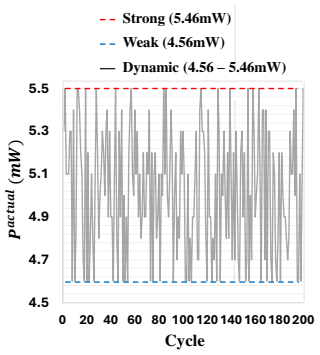


Fig. 10: Harvesting behavior

accuracy with partial adaptability. For three and four tasks, LEAF^{adapt} respectively improved accuracy to 71.5% and 67.5%, compared to 63.5% for the static minimum-width model, but ME failed to execute due to memory overflow caused by the additional classifiers. On the MSP430FR5994, after reserving memory for RTOS tasks, I/O buffers, code, and other system components, approximately 150 KB of FRAM remains available for storing model weights. Since ME requires 70 KB per DNN model due to its auxiliary exit classifiers, it becomes infeasible to deploy more than two tasks within the 256 KB memory limit, as shown in Fig. 8. These results show LEAF^{adapt} effectively leverages time budget to increase accuracy and adapts robustly to dynamic energy conditions without requiring additional memory space.

C. Simulation

Simulation setup. To explore a wider range of operating conditions beyond several settings in our real-world experiments, we conduct additional simulations under various energy environments and task configurations. Specifically, experiments were carried out with two to four multi-DNN tasks, where the task periods were incremented in 10-second steps to represent increasing computational workloads, as shown in Fig. 9 (e.g., a pair of periods [20 and 30], ..., [50 and 60] in seconds, for a set of two tasks). For each configuration, we execute 3000 jobs. We follow/emulate the real-world settings, where $P_i^{\max}$ is set to 9.71 mW, and three different instantaneous harvesting powers P^{actual} based on $P^{\min} = 4.56$ mW: (i) *weak* and *strong* fixed power where P^{actual} is fixed to $P^{\min}$ and to the maximum observed harvesting rate $P^{\text{strong}} = 5.46$ mW, respectively, and (ii) *dynamic* power where P^{actual} randomly varies in $[P^{\min}, P^{\text{strong}})$ as shown in Fig. 10. In particular, *dynamic* power captures the variability typically observed in practical energy-harvesting systems, and evaluates how well LEAF^{adapt} responds to unstable energy conditions.

Simulation results. The fixed-power simulation results align with real-world observations, showing that LEAF^{adapt} consistently outperforms static-width models while maintaining no deadline miss, even for a set of three or four tasks where ME fails due to memory overflow. Note that for a set of two tasks, there exist a few cases where ME achieves higher accuracy than LEAF^{adapt}. This occurs because LEAF^{adapt} selectively activates only the most informative channels, occasionally omitting useful but non-critical features under tight timing constraints. In contrast, ME processes all features up to the selected exit point, achieving higher accuracy in some cases despite its coarse-grained control and memory overhead.

LEAF^{adapt} effectively leverages longer task periods (i.e., less workload) to allocate wider configurations and improve accuracy. For instance, under the strong power setting, in a two-task setting with [20 and 30] periods in seconds, LEAF$^{0.25}$ and LEAF^{adapt} achieved 62% and 65% accuracy, respectively. However, with [50 and 60] in seconds, the accuracy of LEAF$^{0.25}$ remains constant, while LEAF^{adapt} improves to 72.4% by dynamically increasing layer widths, as static models are unable to take advantage of additional runtime due to their fixed-width configurations. This limitation can be further understood by examining the schedulability of different fixed-width settings. LEAF$^{0.25}$ is always schedulable as it corresponds to the minimal configuration. LEAF$^{0.5}$ and LEAF$^{0.75}$ are schedulable only in a subset of task sets, while LEAF$^{1.0}$ is schedulable in only one case due to its long execution time exceeding most deadline constraints. We also observe that LEAF^{adapt} yields higher accuracy under the strong power setting, effectively utilizing shorter power-off phases and longer power-on phases to execute more layers. Under dynamic power conditions, we observe similar trends.

VII. Conclusion

We propose LEAF for real-time multi-DNN inference under intermittent energy conditions. LEAF enables *layer-wise dynamic width reconfiguration* during runtime, allows the system to exploit energy and timing slack at a fine granularity, and finally achieves both timing/energy guarantees and accuracy maximization. We limit our generalization claims to channel-width-scalable CNNs, where inference cost can be adjusted via width control. Extending the framework to architectures with different scaling dimensions is left for future work.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2024-00438248, RS-2025-17622972, RS-2025-22502968). This work was also supported by the Yonsei University Research Fund of 2025-22-0408.

References

[1] J. Hester and J. Sorber, "Flicker: Rapid prototyping for the batteryless internet-of-things," in *ACM Conference on Embedded Network Sensor Systems (Sensys)*, 2017.

[2] D. Kim, J. Ahn, H. Lee, and H. Cha, "Highly responsive batteryless system for indoor light energy harvesting environments," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2023.

[3] M. Arita, Y. Nakamura, S. Ishida, and Y. Arakawa, "Zel: Net-zero-energy lifelogging system using heterogeneous energy harvesters," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2022, pp. 172–179.

[4] A. Montanari, M. Sharma, D. Jenkus, M. Alloulah, L. Qendro, and F. Kawsar, "eperceptive: energy reactive embedded intelligence for batteryless sensors," in *ACM Conference on Embedded Network Sensor Systems (Sensys)*, 2020.

[5] Z. Zhang, C. Liu, and H. Kim, "MII: A multifaceted framework for intermittence-aware inference and scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[6] B. Islam and S. Nirjon, "Scheduling computational and energy harvesting tasks in deadline-aware intermittent systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

[7] K. Wang and Q. Deng, "Mixed-criticality scheduling of energy-harvesting systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022.

[8] S.-T. Cheng, W. S. Lim, C.-H. Tu, and Y.-H. Chang, "Train: A reinforcement learning based timing-aware neural inference on intermittent systems," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023.

[9] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[10] C.-C. Lin, C.-Y. Liu, C.-H. Yen, T.-W. Kuo, and P.-C. Hsiu, "Intermittent-aware neural network pruning," in *ACM/IEEE Design Automation Conference (DAC)*, 2023.

[11] S. Islam, S. Zhou, R. Ran, Y.-F. Jin, W. Wen, C. Ding, and M. Xie, "Eve: Environmental adaptive neural network models for low-power energy harvesting system," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2022.

[12] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *International Conference on Pattern Recognition (ICPR)*, 2016.

[13] Texas Instruments, "MSP430FR5994 16 MHz Ultra-Low-Power Microcontroller," http://www.ti.com/product/MSP430FR5994, 2021.

[14] J. Yu and T. Huang, "Universally slimmable networks and improved training techniques," in *International Conference on Computer Vision (ICCV)*, 2019.

[15] J. Yu, L. Yang, N. Xu, J. Yang, and T. S. Huang, "Slimmable neural networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[16] Y. Wu, Z. Wang, Z. Jia, Y. Shi, and J. Hu, "Intermittent inference with nonuniformly compressed multi-exit neural network for energy harvesting powered devices," in *IEEE Design Automation Conference (DAC)*, 2020.

[17] B. Islam and S. Nirjon, "Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, 2020.

[18] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations (ICLR)*, 2020.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *International Conference on Neural Information Processing Systems (NIPS)*, 2012.

[20] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[21] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *International Conference on Learning Representations (ICLR)*, 2015.

[22] Y. Abdeddaïm, Y. Chandarli, R. I. Davis, and D. Masson, "Schedulability analysis for fixed priority real-time systems with energy-harvesting," in *International Conference on Real-Time Networks and Systems (RTNS)*, 2014.

[23] D. Kang, S. Lee, H. S. Chwa, S.-H. Bae, C. M. Kang, J. Lee, and H. Baek, "RT-MOT: Confidence-aware real-time scheduling framework for multi-object tracking tasks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[25] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[26] R. Barry, "FreeRTOS: Real-time operating system for embedded devices," https://www.freertos.org/, 2024.