2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS) | 978-1-6654-0386-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/RTAS52030.2021.00020

Budget RNNs: Multi-Capacity Neural Networks to Improve In-Sensor Inference Under Energy Budgets

Tejas Kannan
Department of Computer Science
University of Chicago
Chicago, USA
tkannan@uchicago.edu

Henry Hoffmann
Department of Computer Science
University of Chicago
Chicago, USA
hankhoffmann@cs.uchicago.edu

***Abstract*—Recurrent neural networks (RNNs) are well-suited to the sequential inference tasks often found in embedded sensing systems. While RNNs have displayed high accuracy on many tasks, they are poorly equipped for inference under energy budgets that are unknown at design time. Existing RNNs meet energy constraints in sensor environments by training models to subsample input sequences. The tight coupling between the sampling strategy and the RNN prevents these systems from generalizing to new energy budgets at runtime. To address this problem, we present a novel RNN architecture called the *Budget RNN*. Budget RNNs use a leveled architecture to decouple the sampling strategy from the RNN model, allowing a single Budget RNN to change its subsampling behavior at runtime. We further propose a runtime feedback controller to optimize the model's accuracy for a given energy budget. Across a set of budgets, the Budget RNN inference system achieves a mean accuracy of roughly 3 points higher than standard RNNs. Alternatively, Budget RNNs can achieve comparable accuracy to existing RNNs while under 20% smaller budgets.**

I. Introduction

Battery-powered sensors face the challenge of a finite lifetime. When the battery is exhausted, devices need maintenance to continue operation, and this can be costly or impractical for sensors located in hard-to-reach places [1, 2, 3, 4]. Thus, these systems are often augmented with recharging capabilities; e.g., wirelessly [5] or through energy harvesting [6, 7, 3, 4]. Either technique induces energy budgets: until recharged, sensors must perform using only previously-stored energy. As these recharging systems exhibit variance in both generated power and recharge availability, the induced energy budgets are unknown at design time [8].

To meet varying energy budgets, sensors must alter their runtime energy consumption, which typically goes into three tasks: collecting, processing, and communicating data. By collecting data less often—i.e., *sampling* the data—devices reduce the energy consumed by both sensing hardware and subsequent processing [9, 10, 11]. Reducing the data collection rate, however, comes at a cost in error. By collecting fewer values, the sensor creates a less-complete view of the target environment. With this tradeoff, a logical goal of budgeted computation is to minimize the error (or maximize accuracy) while altering energy consumption to meet the constraint.

Understanding how data collection affects error requires understanding sensor computations. Sensors often perform local inference for improved reactivity, privacy, and energy-efficiency [12, 13]. Furthermore, this processing is increasingly focused on deep neural networks (DNNs) [14, 15, 16, 17] due to their high accuracy for image [18] and audio [19, 20] tasks. As many sensing tasks operate on data streams, recurrent neural networks (RNNs) [21, 22] are common DNN models for in-sensor inference [15, 23, 24].

RNNs operate on data sequences of arbitrary length, making them well-suited to sampling techniques, and thus, operating under energy budgets. The challenge involves determining how to minimize the RNN's inference error under an energy constraint. This challenge leads to the concrete problem of selecting the sampling technique that both minimizes the RNN error and conserves enough energy to meet the budget. As we assume the exact energy constraint is not known at design time, the system must have sufficient flexibility to provide high accuracy across a range of energy budgets.

There exist prior RNN solutions—specifically, Phased RNNs [25] and Skip RNNs [26]—that appear to fit into the framework of budgeted computation. While they differ in specifics, these approaches jointly train both an RNN and a sampling strategy. The joint training means that each RNN operates on a single energy consumption level. To meet an energy budget that is only known at runtime, the inference system must use many RNNs, each trained for a different budget. The need to produce many separate networks with different sampling strategies leads to a high training cost. Furthermore, deploying many neural networks will exhaust the memory capacity of low-power devices. An additional downside is the inefficient use of energy budgets. To meet the budget $B \sim [B_{min}, B_{max}]$, the system may have to select an RNN with energy consumption b that is strictly less than the budget. Thus, the system will have a surplus of $B - b > 0$ joules. The goal of this paper is to produce budgeted inference designs that are low overhead (in training time and memory footprint) yet use the entire energy budget to maximize accuracy, rather than waste it as would be done with prior work.

Therefore, we propose a new type of RNN called the Budget RNN. Budget RNNs are designed from the ground-up for accurate inference under energy budgets only known at runtime. The model has a *leveled* architecture in which each level processes a subsequence of inputs. Budget RNNs conserve energy by allowing execution to halt after each subsequence. This design enables a single Budget RNN to dynamically alter its energy consumption. As Budget RNNs support non-contiguous subsequences, these features come without compromising sampling flexibility. We design a runtime controller that uses trainable halting signals from the Budget RNN. This controller optimizes the model accuracy while adhering to an energy budget. The controller generalizes to new budgets and adapts the system to changes in the runtime environment. Furthermore, the combination of early-stopping and dynamic sampling can be realized in a relatively small number of Budget RNNs, leading to reduced training time and low memory overhead.

We evaluate the proposed system in both a simulated environment and a hardware implementation. We show that, across seven datasets, the Budget RNN system achieves a mean accuracy that is 1.5 points higher than that of Skip RNNs [26] and 3 points higher than that of standard RNNs. Alternatively, Budget RNNs can be used to reduce energy needs (and the associated battery and charging capacity). Specifically, for the same accuracy of baseline approaches, Budget RNNs reduce energy requirements by 20%. With respect to Skip RNNs, these accuracy benefits come at no cost in training time.

In summary, this paper makes the following contributions:

- We establish a framework for maximizing inference accuracy under energy budgets. We use this framework to formalize a goal for inference systems that adapt to unseen budgets at runtime.
- We present a novel RNN architecture, called the Budget RNN, for performing inference under energy budgets that are unknown at design time. Budget RNNs change their energy consumption by using a leveled architecture to process subsamples of input sequences. This design achieves better accuracy under energy budgets when compared to existing RNN solutions.
- We design an optimization procedure to control the subsampling behavior of Budget RNNs. This optimizer accounts for energy constraints and provides results that generalize to unseen budgets.
- We create a controller for Budget RNNs to ensure the model meets a given energy constraint. This controller uses a dynamic setpoint that adapts based on feedback from the Budget RNN. The control policy further adapts the system to changes in the runtime environment.

II. Background and Goals

In this section, we motivate the problem of inference under runtime energy budgets using examples from rechargeable sensors (§II-A), provide background information on RNNs (§II-B), establish a formal framework for budgeted inference (§II-C), and identify limitations in prior work (§II-D).

A. Examples of Inference under Energy Budgets

Two examples of target systems are wireless rechargeable sensor networks (WRSNs) [5] and devices with energy harvesting units [4]. We describe these applications below.

WRSNs use a mobile charging unit to wirelessly recharge sensors [5]. The charging unit uses a protocol, such as on-demand charging, to determine when to visit devices [27]. In on-demand charging, sensors notify the mobile charging unit when they are low on power, and various policies determine the exact recharge time [27, 28]. This scenario creates an energy budget: once sensors request a recharge, they have finite remaining energy to use before the charging unit arrives. Furthermore, the time until a recharge is dependent on both the state of the network and the visitation policy. Thus, sensors only know the energy budget at runtime.

Energy harvesting systems supplement batteries with renewable energy. For example, wildlife tracking [1, 4, 6, 8], bridge monitoring [7], and ecosystem observation [2, 3] sensors use solar panels. Energy harvesters have varying performance due to a dependence on their environment; e.g., solar panels generate up to six times less power in overcast rather than sunny weather [4]. This variance creates inherent unreliability in sensor lifetime, which is a problem for operators. For example, ZebraNet sensors desire 72 hours of operation on only battery power [4]. This design requirement leads to energy budgets; given only the energy stored before losing renewable power, sensors must meet the desired uptime. In particular, the residual battery charge before a loss of renewable power is unknown at design time. Thus, such sensors must handle energy constraints that are not known until runtime.

B. Recurrent Neural Networks and Sequential Classification

Sensors collect periodic measurements of their surrounding environment, creating a temporal data stream [29]. As an example, consider human activity recognition [30]. In this task, sensors measure acceleration values at regular intervals. The corresponding inference model uses a sequence of measurements to predict the activity.

Recurrent Neural Networks (RNNs) are a popular model for inference on sequences. RNNs' key feature is their maintenance of an internal *memory state* [21, 22, 31]. At each step, this memory state represents a summary of the inputs observed thus far. RNNs update the memory state using a trainable transition function called the RNN *cell*.

We formally describe RNNs by considering an ordered sequence $\boldsymbol{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_{T-1}\}$. Each vector $\boldsymbol{x}_t \in \mathbb{R}^n$ represents the input measurement at step $t \in [T]$[1]; e.g., for human activity recognition, each $\boldsymbol{x}_t$ holds 3D acceleration values. At step $t \in [T]$, the RNN updates the memory state $\boldsymbol{s}_t \in \mathbb{R}^d$ using the transition function (cell) $S_{\boldsymbol{\vartheta}} : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^d$,

$$\boldsymbol{s}_t = S_{\boldsymbol{\vartheta}}(\boldsymbol{x}_t, \boldsymbol{s}_{t-1}) \quad \forall t \in [T] \tag{1}$$

There exist many RNN cells, from single-layer networks [31] to more complex designs for learning long-term relationships [20, 21, 24, 32, 33].

[1]We use the notation $[N] = \{0, 1, \dots, N-1\}$

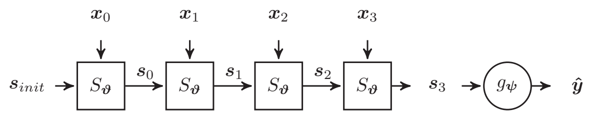

Fig. 1: An RNN on a sequence with three elements. S_{ϑ} represents the RNN cell and g_{ψ} is the readout layer.

RNNs form predictions using a trainable readout function g_{ψ}, which often takes the form of a multi-layer perceptron [34]. The readout layer uses a summary of the input sequence to form the prediction. There are many possible ways to form this summary; in the simplest case, the readout function uses the final memory state s_{T-1} and the prediction is $\hat{y} = g_{\psi}(s_{T-1})$. Figure 1 shows an example of an RNN.

C. Inference under an Energy Budget

We now formalize the goal of designing an RNN-based inference system that achieves low error under runtime energy constraints. Let $\mathcal{F} = \{f_{\theta_1}, f_{\theta_2}, \dots, f_{\theta_p}\}$ be a family of (sequential) inference models. We assume each model incurs a different average energy cost to process a single sequence. Let $\mathcal{X} = \{X^{(0)}, X^{(1)}, \dots, X^{(M-1)}\}$ be a set of M non-overlapping sequences. Each sequence $X^{(m)} = [x_1^{(m)}, x_2^{(m)}, \dots x_T^{(m)}]$ contains T elements where consecutive measurements are sampled Δt seconds apart.

We define an energy constraint as a tuple (B, M) capturing the goal of performing inference on M sequences with energy at most B. We assume that M is a finite integer in $[M_{min}, M_{max}]$ and $B \sim U([B_{min}, B_{max}])$. The ranges describe the set of feasible constraints. Given these constraints, we want a policy π which selects a model in $\mathcal{F}$ at each step $m \in [M]$. The selected model, denoted by $f_m^{\pi} = \pi(\mathcal{F}, X^{(m)}, B, M)$, performs inference on the m^{th} sequence $X^{(m)}$. The dependence of π on B and M arises because the system must adapt to new constraints at runtime.

We want to construct a policy π and a family of inference models $\mathcal{F}$ to (1) minimize the expected error over all possible budgets and (2) meet all energy constraints. Thus we have the following optimization problem (where y_m is the true label for the m^{th} sequence):

$$\pi, \mathcal{F} = \underset{\tilde{\pi}, \tilde{\mathcal{F}}}{\arg\min}\, E_{M,B}\left[\frac{1}{M}\sum_{m=0}^{M-1} \text{Error}(f_m^{\tilde{\pi}}(X^{(m)}), y_m)\right] \quad (2)$$

$$s.t. \quad \sum_{m=0}^{M-1} \text{Energy}(f_m^{\tilde{\pi}}, X^{(m)}) \le B \quad \forall M, B \quad (3)$$

It is infeasible to solve this problem exactly. Even if we could, at design time we do not know the energy budgets.

D. Adaptive Sampling in Recurrent Neural Networks

A family $\mathcal{F}$ for budgeted inference must contain inference models with varying energy costs. Thus, to build a family $\mathcal{F}$ with RNNs, we need a method to control the energy consumption of RNNs during inference. In sensor environments, RNNs can adjust their energy consumption by varying the number of inputs. For example, rather than using a full sequence $[x_0, \dots, x_{T-1}]$ to form a prediction, an RNN can save energy by instead using the subsequence $[x_{\alpha_0}, \dots, x_{\alpha_{r-1}}]$ where $r < n$. Skipping inputs saves energy by reducing the frequency of both processing and collection. These savings are significant when compared to the energy required to communicate predictions to a centralized server. Between the tasks of collection, processing, and communication, data processing has the lowest energy cost. For example, both the TI MSP430 FR5994 [35] and the Atmel SAM L21 [36] draw under 200μA per MHz. In contrast, sensing hardware can consume an order of magnitude more power, and the cost of sensing can even exceed that of radio modules [37]. As transmission only occurs once per sequence, the relative cost and frequency of collection allow subsampling to yield significant energy savings [37]. We can thus create a budgeted inference system with a family of models $\mathcal{F}$ composed of RNNs that subsample input sequences.

Two state-of-the-art RNN architectures support subsampling: Skip RNNs [26] and Phased RNNs [25]. Skip RNNs use a trainable binary gate to skip sequence elements. This gate is jointly trained with the RNN parameters, and it gives the model fine-grained control over the sampling strategy. Phased RNNs use a periodic phase gate to control when the RNN makes updates to its memory state. By unifying the phase gate across all state dimensions, Phased RNNs can use this gate to skip inputs. Thus, both Skip and Phased RNNs can create a family of inference functions ($\mathcal{F}$) by sub-sampling the sequence with varying granularity.

Both approaches train their sampling strategy at the same time they train their RNN parameters. This joint training reduces the flexibility to adapt to runtime energy availability and is a bad match for the constraints of low-power sensing systems. In particular, a Skip or Phased RNN cannot change its sampling strategy after training. This property means that an instance of either RNN operates at only one energy consumption level. To meet energy budgets that are unknown at design time, a system based on either model would need many trained neural networks. This strategy requires a longer training time, and memory restrictions cap the number of deployed models. Furthermore, a reasonable selection policy (π) is to always choose the best RNN that meets the energy constraint. As there are a small, discrete number of models (due to memory constraints on embedded devices), this policy is inefficient; for a constraint (B, M), the best model may use only $b < B$ joules to classify M sequences. Our goal is to design a practical approach that uses the remaining $B - b$ joules to produce more accurate inference results.

III. Budget RNN Architecture

Budget RNNs are a family of RNN architectures designed to (1) deliver high accuracy under energy constraints and (2) generalize to budgets that are unknown until runtime. There exists a tension between these two guiding principles. On one hand, RNN accuracy depends on the sequence subsampling algorithm. As discussed for Skip and Phased RNNs, however, creating a more flexible subsampling algorithm through joint training limits the ability to generalize to new budgets.

Budget RNNs address this tension using four features. First, Budget RNNs have a *leveled architecture* where each level processes a distinct subsequence (§III-A). The Budget RNN produces a prediction after each subsequence. Second, the model avoids recomputation by *merging* RNN memory states across subsequences (§III-B). The leveled architecture allows a single Budget RNN to operate at many energy levels. In particular, controlling the number of executed subsequences determines the number of input elements the Budget RNN uses. This control is possible due to the third feature of Budget RNNs: *halting signals* (§III-C). For each subsequence, Budget RNNs produce a signal which indicates whether the model should halt its execution. By setting thresholds on these signals, an inference policy (π) can dynamically alter the Budget RNN's sampling granularity. These signals decouple the subsampling strategy from the Budget RNN parameters. We discuss this control policy in Section IV. Finally, Budget RNNs are trained with a novel *loss function* that balances the predictions and halting signals across all subsequences (§III-D). Figure 2 shows two Budget RNN architectures.

A. Leveled Architecture

Each Budget RNN level applies a shared RNN to a distinct subsequence. Budget RNNs sequentially process these subsequences, producing a prediction from each. These predictions allow Budget RNNs to exit early during inference. Further, the Budget RNN can skip inputs as the data from unprocessed subsequences does not need to be collected. Thus, Budget RNNs can save on both computation and data collection.

We describe this process formally by considering the subsequence $\boldsymbol{X}^{(\alpha)} = \{\boldsymbol{x}_{\alpha_0}, \ldots, \boldsymbol{x}_{\alpha_{r-1}}\}$ where r is less than the original sequence length T. Similar to standard RNN update (equation 1), Budget RNNs use the state transition model $\mathcal{S}_{\boldsymbol{\vartheta}}$ and readout layer $g_{\boldsymbol{\psi}}$ to perform on the subsequence $\boldsymbol{X}^{(\alpha)}$:

$$\boldsymbol{s}_{\alpha_t} = \mathcal{S}_{\boldsymbol{\vartheta}}(\boldsymbol{x}_{\alpha_t}, \boldsymbol{s}_{\alpha_{t-1}}) \quad \forall t \in [r] \tag{4}$$

$$\hat{\boldsymbol{y}}^{(\alpha)} = g_{\boldsymbol{\psi}}(\boldsymbol{s}_{\alpha_{r-1}}) \tag{5}$$

The transition model $\mathcal{S}_{\boldsymbol{\vartheta}}$ can be any RNN cell [21, 32, 33]. Both $\mathcal{S}_{\boldsymbol{\vartheta}}$ and $g_{\boldsymbol{\psi}}$ are shared across all subsequences, a key feature for reducing memory overhead.

Budget RNNs apply this recurrent process to each subsequence. Subsequence formation is based on two parameters:

- *Number of Subsequences* (L): Each subsequence results in a prediction, so this parameter also represents the number of predictions. We also refer to L as the number of levels—the subsequences form *levels* in the model.
- *Stride Length* (K): This parameter determines the gap between elements in each subsequence. For example, in a sequence of length $T = 6$, the subsequences for a stride length of $K = 2$ are $\{\boldsymbol{x}_0, \boldsymbol{x}_2, \boldsymbol{x}_4\}$ and $\{\boldsymbol{x}_1, \boldsymbol{x}_3, \boldsymbol{x}_5\}$. The ℓ^{th} subsequence has the following form:

$$\boldsymbol{X}^{(\ell)} = \begin{cases} \{\boldsymbol{x}_{(\ell T/L)+n} \mid n \in [\frac{T}{L}]\} & \text{if} \quad K = 1 \\ \{\boldsymbol{x}_{\ell+nK} \mid n \in [\frac{T}{K}]\} & \text{if} \quad K > 1 \end{cases} \tag{6}$$

  When $K > 1$, we set $L = K$ to avoid the having a single element in multiple subsequences.

Both L and K constitute hyperparameters for Budget RNNs. The stride length in particular has an impact on model accuracy, and we discuss how to set this parameter in §IV-D.

B. Merging Memory States Across Subsequences

To maximize accuracy and avoid energy-wasteful redundant computation, Budget RNNs leverage already-processed sequence elements. When executing on level ℓ, the Budget RNN *merges* the new inputs to improve the prediction from level $\ell - 1$. A key property of this merging is the avoidance of backward dependencies. During inference, a sensor supplies the Budget RNN with measurements collected over time. Further, Budget RNNs conserve energy by skipping subsequences. From these two properties, backward dependencies would require the system to either execute the current step using future data or collect and store elements that may be ignored. The former is impossible, and the latter is energy-inefficient.

When $K = 1$, the Budget RNN looks like a standard RNN with L early-exit points [15]. The model combines information between subsequences using the RNN cell. This combination occurs by using the final memory state of the $\ell - 1^{th}$ level as the initial memory state of the ℓ^{th} level. The left-hand-side of Figure 2 shows an example of this design.

When $K > 1$, the merging is more complex due to the interleaving of subsequences. For example, when $T = 4$ and $K = 2$, the Budget RNN uses subsequences $\{\boldsymbol{x}_0, \boldsymbol{x}_2\}$ and $\{\boldsymbol{x}_1, \boldsymbol{x}_3\}$. In this case, the final element of the first subsequence, $\boldsymbol{x}_2$, occurs after elements in the second subsequence. Thus, using $\boldsymbol{s}_2$ as the initial state of the second subsequence creates a backward dependency. Budget RNNs solve this problem by aligning and merging the memory states from each level. Equations (7) and (8) below show how the merging layer $\mathcal{M}$ interacts with the standard RNN cell in (9). The variables $\boldsymbol{W}^{(\mathcal{M})}, \boldsymbol{U}^{(\mathcal{M})}$ and $\boldsymbol{b}^{(\mathcal{M})}$ are trainable parameters, σ is the sigmoid function, and $\odot$ is the element-wise product.

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}^{(\mathcal{M})}\boldsymbol{s}_{t-K} + \boldsymbol{U}^{(\mathcal{M})}\boldsymbol{s}_{t-1} + \boldsymbol{b}^{(\mathcal{M})}) \tag{7}$$

$$\tilde{\boldsymbol{s}}_{t-1} = \boldsymbol{z}_t \odot \boldsymbol{s}_{t-K} + (1 - \boldsymbol{z}_t) \odot \boldsymbol{s}_{t-1} \tag{8}$$

$$\boldsymbol{s}_t = \mathcal{S}_{\boldsymbol{\vartheta}}(\boldsymbol{x}_t, \tilde{\boldsymbol{s}}_{t-1}) \tag{9}$$

To clarify this design, let step t belong to subsequence ℓ. This merging layer makes $\boldsymbol{s}_t$ dependent on all inputs collected thus far. The state $\boldsymbol{s}_{t-K}$ precedes step t in the ℓ^{th} subsequence and represents a summary of the *current* subsequence up to step t. The state $\boldsymbol{s}_{t-1}$ belongs to subsequence $\ell - 1$ and represents a summary of the already-collected elements in *previous* subsequences. Using a combination of $\boldsymbol{s}_{t-1}$ and $\boldsymbol{s}_{t-K}$ makes $\boldsymbol{s}_t$ dependent on all elements collected up to step t. The right-hand portion of Figure 2 depicts this design.

Within this leveled architecture, the error should always improve as the model observes more data. In practice, however, this desired behavior does not always hold. We hypothesize that this issue stems from two factors: (1) weight sharing across levels and (2) a loss function based on an unweighted

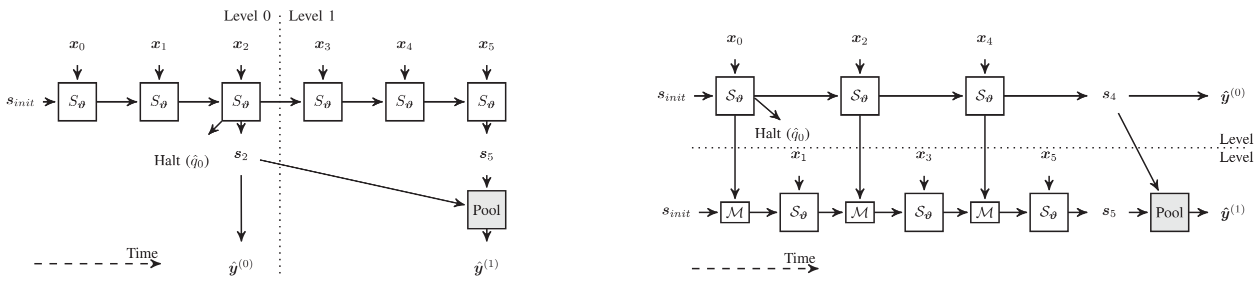

Fig. 2: Budget RNNs with sequence length $T = 6$, $L = 2$ levels, and stride lengths $K = 1$ (left) and $K = 2$ (right).

average of the per-level loss (§III-D). To mitigate this problem, we use an additional output pooling layer to combine the predictions across subsequences. The pooling operation allows Budget RNNs to use predictions from previous levels if doing so lowers the error. As shown in the equations below, this layer uses a trainable weighted average to aggregate predictions. The variable $\boldsymbol{s}^{(\ell)}$ is the final memory state for level ℓ, $\boldsymbol{w}_{pool}, \boldsymbol{u}_{pool}$ and b_{pool} are trainable, and $g_{\boldsymbol{\psi}}$ is the readout function.

$$r_k = \boldsymbol{w}_{pool}^\top \boldsymbol{s}^{(\ell)} + \boldsymbol{u}_{pool}^\top \boldsymbol{s}^{(k)} + b_{pool} \quad \forall k \leq \ell \tag{10}$$

$$\beta_k = Normalize(r_0, \ldots, r_\ell)_k \quad \forall k \leq \ell \tag{11}$$

$$\hat{\boldsymbol{y}}^{(\ell)} = \sum_{k=0}^{\ell} \beta_k g_{\boldsymbol{\psi}}(\boldsymbol{s}^{(k)}) \tag{12}$$

In our implementation, we use a SparseMax normalization [38] instead of a softmax normalization due to better numerical stability when implemented in fixed-point arithmetic on MCUs.

C. Halting Signals

Budget RNN's use early-exit points to alter their energy consumption. For example, to conserve energy, a Budget RNN can stop at an intermediate level ℓ; the sensor then does not need to capture inputs belonging to subsequences $\ell' > \ell$. To maximize accuracy, the inference policy (π) should control the number of executed levels in a data-dependent manner [26]. That is, the system should use inputs to determine when to exit inference. Budget RNNs support the ability to make data-dependent decisions using trainable halting signals [39, 40, 41]. At each level $\ell \in [L]$, these signals indicate the probability the current level's prediction is correct. The policy π controls the halting behavior using thresholds on these signals (§IV).

We describe these halting signals by considering two cases. First, when $K = 1$, the model creates halting signals using the final memory state of each level. Second, when $K > 1$, the Budget RNN creates halting signals from the first element of each subsequence. This discrepancy is a result of the interleaving of elements across subsequences. For example, during inference, the system must decide whether to collect $\boldsymbol{x}_1$ after processing $\boldsymbol{x}_0$. When $K > 1$, $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ belong to different subsequences. Consider if the Budget RNN created the halting output after completing level $\ell = 0$. The system would need to collect elements (such as $\boldsymbol{x}_1$) that may be ignored if the system decides to halt at $\ell = 0$. Such extraneous data collection wastes energy. Thus, to efficiently collect inputs, the system must use $\boldsymbol{x}_0$ to decide whether it should halt at level $\ell = 0$. Therefore, when $K > 1$, the Budget RNN must use the first memory state of the ℓ^{th} subsequence to construct the halting signal for level ℓ.

Budget RNNs use a shared, trainable layer to create the halting signals. The equations below show the halting function $h_{\boldsymbol{\varphi}}$ applied to level ℓ. The term $\boldsymbol{s}_{\delta(\ell)}$ is the final state of level ℓ when $K = 1$ and the first state of level ℓ when $K > 1$. The variables $\boldsymbol{W}_{halt,1}, \boldsymbol{w}_{halt,2}, \boldsymbol{b}_{halt,1}$ and $b_{halt,2}$ are trainable, and ϕ is the nonlinear activation function.

$$\boldsymbol{h}_\ell = \phi(\boldsymbol{W}_{halt,1}\boldsymbol{s}_{\delta(\ell)} + \boldsymbol{b}_{halt,1}) \tag{13}$$

$$\hat{q}_\ell = \sigma(\boldsymbol{w}_{halt,2}^\top \boldsymbol{h}_\ell + b_{halt,2}) \tag{14}$$

Budget RNNs train the halting signal $\hat{q}_\ell$ to predict whether the model is correct at level ℓ. For classification tasks, the label is $q_\ell = 1[y = \arg\max \hat{\boldsymbol{y}}^{(\ell)}]$ where $1[\cdot]$ is an indicator function. For regression tasks, the label is $q_\ell = 1\left[\|\boldsymbol{y} - \hat{\boldsymbol{y}}^{(\ell)}\|_2^2 < \epsilon\right]$ for a threshold $\epsilon > 0$. In either case, the label is a non-differentiable function of the Budget RNN parameters. We address this problem by treating the label as a constant. This treatment is advantageous because it prevents the Budget RNN from crafting predictions to match the halting signals. In particular, this feature avoids behavior where the Budget RNN predicts a sequence wrong *on purpose* to match a halting signal close to zero. Such behavior leads to a poor model.

D. Loss Function

All neural networks are trained to minimize a loss function. This function typically measures the difference between the predicted output and the true result. The novel architecture of Budget RNNs requires a new loss function to balance the optimization of both predictions and halting signals.

At each level $\ell \in [L]$, Budget RNNs produce two outputs: the prediction $\boldsymbol{y}^{(\ell)}$ and the halting signal $\hat{q}_\ell$. We use a loss function to train the model using the outputs from all levels. The loss function below expresses this goal. The function $\tilde{\mathcal{L}}$ is the per-output loss (e.g. cross-entropy, mean-squared error).

$$\mathcal{L}(\{\hat{\boldsymbol{y}}^{(\ell)}\}, \{\hat{q}_\ell\}, y, \{q_\ell\}) = \sum_{\ell=0}^{L-1} \left(\tilde{\mathcal{L}}(\hat{\boldsymbol{y}}^{(\ell)}, y) + \gamma \tilde{\mathcal{L}}(\hat{q}_\ell, q_\ell)\right) \tag{15}$$

Algorithm 1 Adaptive Inference Algorithm for Budget RNNs

```
1: procedure AdaptiveInference($f_{\boldsymbol{\theta}}, \boldsymbol{z}, L, K, T$)
2:     $\ell_{max} \leftarrow L$
3:     $(S_{\boldsymbol{\vartheta}}, \mathcal{M}, g_{\boldsymbol{\psi}}, h_{\boldsymbol{\varphi}}) \leftarrow f_{\boldsymbol{\theta}}$
4:     for $t \in [T]$ do
5:         $\ell \leftarrow$ LevelOf$(t, K, L)$
6:         if $\ell > \ell_{max}$ then
7:             continue
8:         $\boldsymbol{x}_t \leftarrow$ CollectInput()
9:         $\tilde{\boldsymbol{s}}_{t-1} \leftarrow \mathcal{M}(\boldsymbol{s}_{t-1}, \boldsymbol{s}_{t-K})$      ▷ (Eq 8)
10:        $\boldsymbol{s}_t \leftarrow S_{\boldsymbol{\vartheta}}(\boldsymbol{x}_t, \tilde{\boldsymbol{s}}_{t-1})$      ▷ (Eq 9)
11:        $\hat{q}_\ell \leftarrow h_{\boldsymbol{\varphi}}(\boldsymbol{s}_t)$      ▷ (Eq 14)
12:        if isHaltState$(t, K, L)$ and $\hat{q}_\ell \geq \boldsymbol{z}_\ell$ then
13:            $\ell_{max} \leftarrow \ell$
14:        if IsLast$(t, K, L)$ and $\ell = \ell_{max}$ then
15:            return Prediction$(\boldsymbol{s}_t, g_{\boldsymbol{\psi}})$      ▷ (Eq 12)
16:    return Prediction$(\boldsymbol{s}_{T-1}, g_{\boldsymbol{\psi}})$      ▷ (Eq 12)
```

The variable γ determines the relative emphasis on the halting loss. We slowly increase γ over the first few epochs until it reaches γ_0. In our experiments, we set $\gamma_0 = 0.01$. This slow increase improves training because the predictions frequently change in the first few epochs.

IV. Control Policy and Subsampling Algorithm

The Budget RNN output levels compose a family of functions $\mathcal{F}$ for budgeted inference. The inference system needs a selection policy π that minimizes error and adapts to unseen constraints. The Budget RNN policy meets these goals by dynamically selecting the output level in a budget-specific manner. This choice determines the number of collected inputs and the performed computation, ensuring efficient use of the available energy.

The Budget RNN policy achieves this dynamic behavior through four features. First, it performs adaptive inference (§IV-A) by setting thresholds on the Budget RNN's halting signals. Second, an optimizer (§IV-B) tunes these thresholds over multiple potential constraints. The policy uses an interpolation method to generalize the thresholds to unseen budgets. Third, a feedback control system mitigates generalization errors while adapting to unforeseen changes in the runtime environment (§IV-C). Finally, the policy improves accuracy by combining distinct Budget RNNs with different stride lengths (§IV-D).

A. Adaptive Inference Algorithm

The Budget RNN inference policy π uses the halting signals to control the number of consumed inputs; i.e., π uses information from the model to determine the early-exiting behavior. *Thus, the policy adapts the energy consumption based on the input sequence.* These data-dependent decisions allow the policy to conserve energy on sequences where the model needs fewer inputs to achieve a low error. The policy spends this saved energy to reconcile "harder" sequences.

The policy π implements early-exiting using a budget-specific threshold vector $\boldsymbol{z}^{(B,M)}$ where (B, M) is the energy constraint. When the halting probability $\hat{q}_\ell$ at level ℓ is greater than the threshold $\boldsymbol{z}_\ell^{(B,M)}$, the inference terminates at level ℓ. Otherwise, it continues to the next level. Algorithm 1 describes this adaptive inference routine. We emphasize an integral feature of this design: when inference halts at level ℓ, the system does *not* collect measurements associated with levels $\ell' > \ell$. Thus, when halting at lower levels, the Budget RNN exhibits significant energy savings by not engaging the sensing hardware to collect unused measurements.

Algorithm 2 Optimization Algorithm for Halting Thresholds

```
1: procedure FitThresholds($\mathcal{D}, f_{\boldsymbol{\theta}}, L, (B, M)$)
2:     $\boldsymbol{z} \sim U([0,1]^L)$
3:     while not converged do
4:         $k \sim \{0, \ldots, L-2\}$ Uniformly at Random
5:         $\boldsymbol{z}_k \leftarrow \arg\min_{\boldsymbol{z}_k \in [0,1]}$ AdjError$(f_{\boldsymbol{\theta}}, \mathcal{D}, \boldsymbol{z}, B, M)$
6:     return $\boldsymbol{z}$
```

B. Optimizing Halting Thresholds

The policy π controls energy consumption using thresholds on the Budget RNN halting signals. We create these thresholds using an additional training step. This process uses a coordinate descent technique [42] to fit thresholds for a given constraint. The optimizer creates thresholds to both minimize error and meet the budget.

Consider a Budget RNN $f_{\boldsymbol{\theta}}$ and an energy constraint (B, M). The optimization problem below formalizes this goal. The term $\mathcal{D} = \{\boldsymbol{X}^{(i)}, y^{(i)}\}_{i=0}^{M-1}$ is the training dataset, and the "Error" function uses predictions from the adaptive inference routine in algorithm 1. This problem is similar to that of the original problem statement for budgeted inference (Eq 3).

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z} \in [0,1]^L} \sum_{i=0}^{M-1} \text{Error}(f_{\boldsymbol{\theta}}, \boldsymbol{z}, \boldsymbol{X}^{(i)}, y^{(i)}) \qquad (16)$$

$$s.t. \quad \sum_{i=0}^{M-1} \text{Energy}(f_{\boldsymbol{\theta}}, \boldsymbol{z}, \boldsymbol{X}^{(i)}) \leq B \qquad (17)$$

This optimization problem is hard to solve for three reasons. First, the runtime energy consumption is often unknown at design time. We thus approximate the energy consumption using profiled values from sensing hardware. Second, the optimization problem may have a discontinuous objective function. For example, in classification tasks, the error function is the negative system accuracy. To address this challenge, we use a coordinate descent solver. At each step, the optimizer selects a random threshold index and finds its optimal value in $[0, 1]$. Finally, the optimizer must adhere to the energy constraint. We implement this behavior by forming an adjusted error function that penalizes budget violations. The equations below show an adjusted error function based on accuracy. This function scales the accuracy using the number of sequences that fit under the budget. The variable $\hat{b}$ is the average energy per sequence, and the "Acc" function uses predictions from the adaptive inference routine in algorithm (1).

$$\hat{a}(f_{\boldsymbol{\theta}}, \mathcal{D}, \boldsymbol{z}, B, M) = -\text{Acc}(f_{\boldsymbol{\theta}}, \boldsymbol{z}, \mathcal{D}) \cdot \frac{\min(M, B/\hat{b})}{M} \qquad (18)$$

Algorithm 3 Budget RNN Controller Policy

```
1: procedure ControlPolicy($f_{\boldsymbol{\theta}}, (B, M), L, K, T$)
2:     $Y \leftarrow [\ ]$
3:     $B_0 \leftarrow B$
4:     for $m \in [M]$ do
5:         $\boldsymbol{z} \leftarrow$ InterpolateThresholds($B_m, M$)
6:         $\hat{y}_m \leftarrow$ AdaptiveInference($f_{\boldsymbol{\theta}}, \boldsymbol{z}$)        ▷ (Alg 1)
7:         $b_{obs} \leftarrow$ ObserveEnergy()
8:         $(b_l, b_u) \leftarrow$ GetSetpoint($B, M, m$)        ▷ (Eq 21)
9:         $e \leftarrow$ PIDControlError($b_{obs}, (b_l, b_u)$)
10:        $B_{m+1} \leftarrow B + e$        ▷ $e > 0$ means $b_{obs}$ is low
11:        $Y \leftarrow Y \cup [\hat{y}_m]$
12:    return $Y$
```

Algorithm 2 describes the optimizer. Below we describe a few implementation details regarding the halting thresholds.

- *Initialization Strategy*: We initialize the threshold for level ℓ using $U([\tilde{q}_\ell, 1])$ where $\tilde{q}_\ell$ is the median halting signal. We further set thresholds to zero for levels that violate the budget on a per-step basis. This strategy creates thresholds that approximately meet the budget, avoiding cases where the optimizer makes suboptimal decisions just to meet the constraint early in training.
- *Coordinate Descent Step*: The key step during each iteration involves finding the optimal threshold. We find this value approximately by sweeping over a quantized subset of [0, 1]. In our implementation, we search over $\{0, 1\} \cup \{2^{n-k} \mid n \in [k]\}$ for $k = 256$.
- *Convergence Detection*: We detect convergence by assessing how thresholds generalize to unseen validation data. Following standard practice, the optimizer terminates after Q iterations of non-improved validation error [43]. This early-stopping mitigates the risk of overfitting. We set $Q = 25$ in our experiments.
- *Population-based Training*: To mitigate sensitivity to initialization, we use a population-based approach [44] to fit many threshold vectors in parallel. Every R iterations, underperforming thresholds are set to the best vector. These copied thresholds are randomly perturbed to explore promising regions of the solution space. We use $R = 10$ in our experiments.
- *Generalizing to Unseen Budgets*: The inference policy must adapt to unseen energy budgets. The policy performs this adaptation by creating new thresholds at runtime. For an unseen budget, the policy finds the two nearest known budgets that bound this new constraint. The policy creates new thresholds by linearly interpolating the thresholds of the bounding budgets. In general, the relationship between thresholds and energy consumption may be nonlinear. We nevertheless find linear interpolation to be a low-overhead heuristic that empirically performs well. To mitigate interpolation errors, we use a runtime controller adjust the selected thresholds (§IV-C).

C. Runtime Controller

The inference policy for Budget RNNs uses budget-specific halting thresholds to minimize error. Alone, the threshold training process has two qualities that hurt the system's ability to generalize. First, the optimizer approximates energy consumption using profiled values. The thresholds may yield a suboptimal error if the runtime and profiled environments differ. Second, the threshold interpolation method only approximately meets unseen budgets. The inference policy remedies these issues using a PID controller [45]. For each sequence, the controller compares the observed energy consumption with the expected energy based on training. The policy uses the control error to adjust the halting thresholds by changing the budget used during interpolation. For example, if energy consumption is lower than expected, the controller will use thresholds based on a larger budget. This controller thus maps the runtime environment into the space constructed during training. Algorithm 3 describes this process. In our implementation, we update the budget every $W = 20$ steps.

The main challenge associated with this controller involves creating the setpoint, which is nontrivial because the adaptive inference algorithm may not evenly spread the available energy across all M sequences. To account for system variation, the controller uses a dynamic setpoint based on a confidence bound. This bound is based on the expected energy consumption and the corresponding estimator variance.

We derive this setpoint by considering b_k to be the average energy consumed when the Budget RNN predicts the label $k \in [C]$. For each label, the energy consumption may vary due to the adaptive inference routine (Alg 1). We thus assume that b_k is a normally distributed random variable. Consider when the system has processed $m < M$ inputs. We define r_k to be the number of sequences with label k and $r_k^{(m)}$ to be the number of sequences in class k that occur in the first m sequences. Let $X_{0:m}$ and $X_{m:M}$ be random variables representing the energy consumption in the first m and final $M-m$ steps respectively. The controller creates a setpoint based on $E[X_{0:m}]$. Given the total energy budget B, we want to have $B = E[X_{0:m}] + E[X_{m:M}]$. This relationship expresses the goal that the system should use the entire budget. Evaluating $E[X_{m:M}]$ yields the following expression for $E[X_{0:m}]$.

$$E[X_{0:m}] = B - \sum_{k=0}^{C-1} (r_k - r_k^{(m)}) E[b_k] \tag{19}$$

We estimate the values for r_k, $r_k^{(m)}$, and $E[b_k]$ using both the training set and the profiled energy values. We update these distributions with values obtained at runtime. These approximations result in an estimator $\hat{E}[X_{0:m}]$ for the expectation.

A setpoint based on this expected value alone does not capture the variance associated with Budget RNN execution. We instead use a confidence bound [46] which accounts for the estimator variance. Under the assumption that r_k is deterministic and the b_k variables are independent, we obtain the following equation for the variance of $X_{0:m}$. In practice,

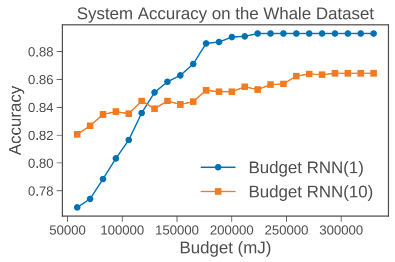

Fig. 3: The accuracy for two distinct Budget RNNs on the task of classifying Whale sounds [49].

we again use an estimator of $\text{Var}[X_{0:m}]$ based on estimates of r_k, $r_k^{(m)}$ and $\text{Var}[b_k]$ from the training set.

$$\text{Var}[X_{0:m}] = \sum_{k=0}^{C-1} (r_k - r_k^{(m)})^2 \text{Var}[b_k] \tag{20}$$

From the Fisher information for Gaussian random variables, the variance of the estimator $\hat{E}[X_{0:m}]$ converges to $\frac{1}{m}\text{Var}[X_{0:m}]$ [47, 48]. With these estimations, we construct the following setpoint for the PID controller.

$$\text{Setpoint}(m) = \left(\hat{E}[X_{0:m}] \pm \sqrt{\frac{1}{m} \hat{\text{Var}}[X_{0:m}]} \right) \tag{21}$$

When the observed energy is within this confidence bound, the controller error is zero. Otherwise, the control error is set based on the difference to the nearest side of the bound.

We highlight one important aspect of this design: the controller does *not* need the budget at design time. At runtime, the controller uses Equations (19) and (21) to dynamically construct a setpoint for a given budget (B, M) . The system uses this setpoint to adapt the Budget RNN halting thresholds and meet the energy constraint (Algorithm 3).

D. Budget RNN Selection

The presented inference policy uses thresholds to control a single Budget RNN. On many datasets, we observe that one Budget RNN is not enough to deliver high accuracy across all budgets. This trend occurs due to the impact of the stride length parameter (K). Importantly, when comparing Budget RNNs with the stride length K_1 and K_2, the model with K_1 may be better for some budgets and worse for others. Figure 3 shows such an example. To achieve the best performance across all budgets, the policy should leverage both models.

We use this insight to build the family $\mathcal{F}$ with many distinct Budget RNNs. For a given budget, the policy π first selects the Budget RNN based on validation accuracy. The policy proceeds to use the runtime controller and corresponding thresholds for the chosen Budget RNN. We observe that providing the policy with a few models yields better performance. Further, this dynamic selection reduces the need to search over stride lengths to find the *single* best model.

Using many distinct RNNs has the downside of increasing the system's memory footprint. In our experiments, we use only two Budget RNNs to achieve high accuracy over a spectrum of budgets. This modest number meets any reasonable memory constraints imposed by low-power devices.

V. Evaluation

We evaluate the Budget RNN's ability to support accurate inference under energy budgets on embedded sensing systems. We compare to standard RNNs as well as the current state-of-the-art RNNs that support sub-sampling (Phased [25] and Skip [26] RNNs). We compare these systems across many datasets and energy budgets in a simulated environment[2]. We supplement these results with an evaluation on an embedded device. This evaluation shows the following:

1) Under the same budgets, the Budget RNN system achieves higher accuracy than that of baseline systems. Across all datasets, the Budget RNN system has a mean accuracy of 1.5 points greater than Skip RNNs, 2.7 points greater than Phased RNNs, and 3 points greater than standard RNNs (§V-B). Further, Budget RNNs can operate with 20% smaller energy budgets and still achieve accuracy comparable to the baselines (§V-C).
2) The control policy allows the Budget RNN system to adapt to new runtime environments. This adaptive behavior enables Budget RNNs to achieve greater improvements in such settings (§V-D).
3) The adaptive decisions made by the Budget RNN policy are key to the system's performance. Removing the adaptive policy or making randomized decisions results in lower accuracy (§V-E).
4) Budget RNNs have lower training costs when compared to both Skip and Phased RNNs. These baseline systems take over 2.3× longer to train on average (§V-F). Budget RNNs display cheaper training while delivering higher accuracy across all budgets.
5) The higher accuracy of Budget RNNs translates to an embedded system. Across two budgets, Budget RNNs maintain their improvement over Skip RNNs (§V-G).

A. Experimental Setup

1) Baseline Systems: We use three RNN variations to create baseline systems: standard RNNs, Phased RNNs [25], and Skip RNNs [26]. The first baseline uses early-exiting in standard RNNs [15]. This model creates a prediction from each memory state, and we interpret these outputs as models in $\mathcal{F}_{rnn}$. We train the RNN to minimize the average loss across all predictions. The second baseline uses Phased RNNs [25]. Each model in $\mathcal{F}_{phased}$ uses a phase gate with a different open rate. The final baseline uses Skip RNNs [26]. We train each Skip RNN in $\mathcal{F}_{skip}$ to meet a different target number of elements. This target is enforced through an L2 loss term. For both the Phased and the Skip RNN systems, we create systems with 10 distinct models[3]. Using 10 models provides these baselines with a good tradeoff between energy and accuracy while maintaining a reasonable memory footprint.

[2] All code is available at https://github.com/tejaskannan/budget-rnn

[3] We use 8 models on the pen digits dataset due to shorter sequences.

Dataset	Seq Len	Classes	# Train	# Val	# Test
EMG [50]	50	7	10,330	2,887	4,975
FordA [51]	20	2	3,060	541	1,320
Pavement [52]	30	3	33,336	5,815	39,939
Pedestrian [53, 54]	20	10	2,024	364	4,878
Pen Digits [55]	8	10	6,033	1,461	3,498
UCI HAR [56]	50	12	18,229	6,800	10,197
Whale [49, 57]	30	2	9,351	1,583	1,962

Fig. 4: Evaluation dataset characteristics.

Dataset	Bluetooth Energy Profile				Temperature Energy Profile			
	RNN	*Phased*	*Skip*	*Budget*	*RNN*	*Phased*	*Skip*	*Budget*
EMG	0.713	0.716	0.713	**0.724**	0.702	0.704	0.712	**0.721**
Ford A	0.881	0.859	0.876	**0.886**	0.867	0.845	0.860	**0.868**
Pavement	0.715	0.669	0.670	**0.716**	**0.697**	0.642	0.658	0.695
Pedestrian	0.607	0.702	0.714	**0.720**	0.543	0.665	0.649	**0.674**
Pen Digits	0.832	0.831	0.869	**0.881**	0.838	0.842	0.871	**0.879**
UCI HAR	0.862	0.828	0.859	**0.865**	0.848	0.820	**0.856**	**0.856**
Whale	0.864	0.869	0.871	**0.879**	0.849	0.859	0.865	**0.871**
All	0.776	0.778	0.791	**0.806**	0.753	0.761	0.773	**0.788**

Fig. 5: Geometric mean accuracy across all budgets. RNN refers to the standard RNN.

For each baseline, we use a fixed selection policy π_{fixed}. This policy selects the best model whose estimated energy meets a given budget. The policy estimates energy consumption using both profiled values and the average number of processed inputs. For all models that meet the budget, π_{fixed} selects the model with the best validation accuracy.

2) Datasets and Neural Network Training: We evaluate system performance on the seven datasets in Figure 4. Each dataset constitutes a sensor-like classification task. For all RNNs, we use single-layer UGRNN cells [33] with a 20 dimensional state. This design limits the memory footprint of each model. All models use a readout layer with 32 hidden units and a Leaky ReLU [58] activation. We fit the RNNs using stochastic gradient descent [59] with an Adam optimizer [60] and a learning rate of 10^{-4}. We use the same batch size for all models. We train the RNNs in Tensorflow [61] for at most 250 epochs with early stopping after 25 epochs.

3) Simulated Environment: We conduct the majority of this evaluation in a simulated environment that bases energy consumption on profiled values from a TI MSP430 FR5994 MCU [35]. We measure the energy required to capture and process inputs using the EnergyTrace tool [62]. We experiment with two energy profiles. The first uses a DHT-11 temperature sensor, and the second simulates collection through an HM-10 Bluetooth module. When sampled at 0.5Hz, the 3.3V MCU with the DHT-11 consumes about 5.6mJ per input; the HM-10 system uses about 29.6mJ per input. These values represent varying points on the spectrum of sensor energy consumption. The simulator also incorporates the energy required for data processing. This feature accounts for the higher computational cost of Budget RNNs (§V-H). For simplicity, we assume that both Phased and Skip RNNs incur the same processing cost as standard RNNs. This assumption is conservative as Phased and Skip RNNs perform additional computation.

4) Hardware Environment: We supplement the simulated environment with an experiment on the actual TI MSP430 FR5994 MCU [35]. We use an HM-10 BLE module to transmit inputs to the device. We power the MCU using a supercapacitor and vary the budget by setting the total capacitance. The Budget RNN controller obtains energy feedback by measuring the voltage across the capacitor. We quantize the neural network weights to 16-bit fixed-point values and execute the RNNs using the on-board low-energy accelerator.

5) Budget RNN Parameters: In all experiments, we use a Budget RNN system with two distinct models. These models have stride lengths of 1 and 10 respectively[4]. The policy selects the Budget RNN at runtime as described in §IV-D. We fit halting thresholds for 11 budgets on the Bluetooth profile and 12 budgets in the temperature setting. The Budget RNN system automatically generalizes to unseen budgets.

B. Inference Accuracy

For each dataset, we evaluate the inference accuracy on a set of energy budgets in the simulated environment. We set M to the size of each testing set and use energy values B to standardize the ratios B/M across datasets. We use 22 and 26 budgets on the Bluetooth and the temperature profiles respectively. We compare the systems by computing the geometric mean accuracy across all budgets.

On average, the Budget RNN achieves the best accuracy across all datasets and budgets. Figure 5 shows these results for both energy profiles. When compared to the standard RNN, Budget RNNs display a mean accuracy gain of over 3 points. Further, Budget RNNs outperform Skip RNNs (1.5 points) and Phased RNNs (2.7 points). These results hold for both energy profiles. The table further shows how no single baseline delivers high accuracy across all datasets. In contrast, the Budget RNN system displays consistency–it almost matches or outperforms the *best* baseline system on all datasets. In a budget-by-budget comparison, the Budget RNN also shows distinct improvements. Using the Bluetooth energy profile, Budget RNNs have higher accuracy than standard RNNs on 77.3% (119 / 154) of budgets. The same comparison against Phased and Skip RNNs shows higher accuracy on 82.5% (127 / 154) and 83.1% (128 / 154) of budgets respectively. A similar trend holds on the temperature profile.

To better understand the observed accuracy differences, we compare the per-budget accuracy on the Pen Digits task. Figure 6 displays the obtained accuracy values. We observe how the Budget RNN provides distinct benefits under tight budgets. These gains become smaller as the budget increases. In particular, for large budgets, the Budget RNN shows slightly worse accuracy than the standard and Phased RNNs. Given no assumptions about the budget, however, we find that Budget RNNs display the best overall result.

Better budget utilization appears to be a reason for the Budget RNN's improved accuracy. On average, Budget RNNs

[4]We use a stride length of 4 on the Pen Digits task due to shorter sequences.

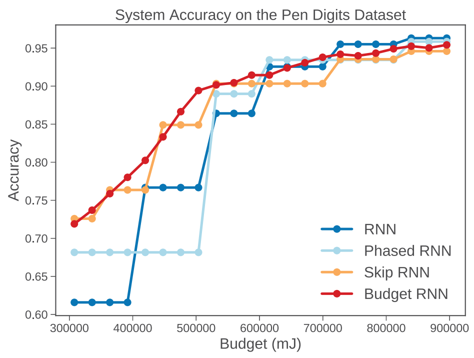

Fig. 6: System accuracy on the Pen Digits dataset using the Bluetooth energy profile.

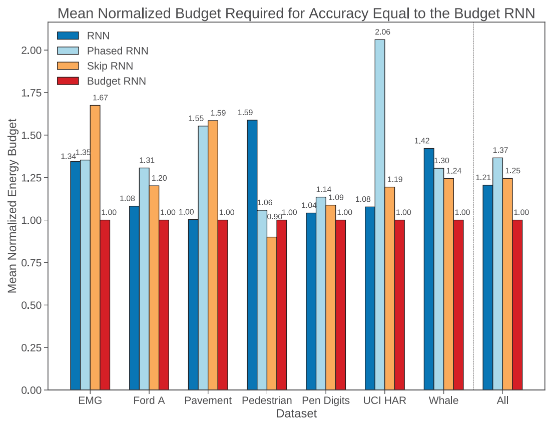

Fig. 7: Geometric mean of normalized budget to obtain accuracy equal to the Budget RNN. Values above one indicate better performance by the Budget RNN.

use 99.2% of the budget, while RNNs (94.7%), Phased RNNs (78.1%), and Skip RNNs (80.4%) display inefficiencies. The Phased and Skip RNNs have lower utilization because the models sometimes show lower accuracy when given more inputs. As this utilization pattern occurs on both the Bluetooth and temperature profiles, these results further demonstrate how the Budget RNN's advantages hold for multiple sensors.

C. Energy Comparison

To place the higher accuracy of Budget RNNs into the context of energy, we estimate the budget needed by Budget RNNs to obtain accuracy equivalent to the baseline systems. For each baseline on budget B, we find the smallest budget B' such that the Budget RNN displays the same accuracy on B' as that of the baseline on B. We use the ratio B/B' to compare the budgets needed to obtain the same accuracy. Figure 7 shows this comparison on the Bluetooth energy profile. On average, the Budget RNN system can use over 20% smaller budgets and still deliver comparable accuracy to the baseline systems. This value describes how the accuracy improvement (Figure 5) manifests itself in terms of energy.

Bias	**Bluetooth Energy Profile**				**Temperature Energy Profile**			
	RNN	*Phased*	*Skip*	*Budget*	*RNN*	*Phased*	*Skip*	*Budget*
+20%	0.664	0.724	0.731	**0.762**	0.603	0.674	0.670	**0.702**
+10%	0.729	0.764	0.778	**0.786**	0.681	0.733	0.738	**0.750**
-10%	0.776	0.778	0.791	**0.813**	0.753	0.764	0.774	**0.797**
-20%	0.776	0.778	0.791	**0.818**	0.753	0.765	0.774	**0.802**
All	0.735	0.761	0.772	**0.794**	0.695	0.733	0.738	**0.762**

Fig. 8: Geometric mean accuracy across all datasets and budgets in settings with a nonzero energy bias.

Dataset	**Bluetooth Energy Profile**			**Temperature Energy Profile**		
	Fixed	*Random*	*Adaptive*	*Fixed*	*Random*	*Adaptive*
EMG	0.722	0.718	**0.724**	0.720	0.718	**0.721**
Ford A	0.874	0.856	**0.886**	0.857	0.844	**0.868**
Pavement	0.706	0.689	**0.716**	0.684	0.673	**0.695**
Pedestrian	0.692	0.624	**0.720**	0.642	0.596	**0.674**
Pen Digits	0.843	0.827	**0.881**	0.843	0.836	**0.879**
UCI HAR	0.860	0.841	**0.865**	0.853	0.841	**0.856**
Whale	0.875	0.852	**0.879**	0.867	0.849	**0.871**
All	0.792	0.767	**0.806**	0.774	0.757	**0.788**

Fig. 9: Geometric mean accuracy of Budget RNN variants. The adaptive results correspond to the full Budget RNN system.

D. Accuracy on Unseen Energy Profiles

The Budget RNN system constructs halting thresholds using profiled energy values. The policy (π) decouples the system from the profiled values using a runtime controller. We evaluate this decoupling by measuring how the system performs in an environment with biased noise. Specifically, in this setting, we model the energy consumption to collect and process r samples as $e(r) = \omega_r + \epsilon$ where ω_r is the profiled energy value and $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$. The bias $\mu \neq 0$ is unknown to the systems, and each policy starts under the assumption of an unbiased environment. Positive biases mean that, at each step, the system consumes more energy than expected. Negative biases work in the opposite fashion.

We evaluate the systems over four biases that represent about $\pm10\%$ and $\pm20\%$ of the profiled energy consumption. Figure 8 shows the geometric mean accuracy across all budgets and datasets. In this setting, Budget RNNs show even higher accuracy relative to the baselines. Across all biases, Budget RNNs show a mean accuracy that is almost 6 points higher than standard RNNs. This improvement is roughly double what we observe in the unbiased setting. The same comparison against Phased RNNs (2.9 points) and Skip RNNs (2.2 points) also yields a larger improvement for Budget RNNs in the biased environment. These benefits are a direct result of adaptivity. The baseline systems use a fixed policy that does not account for differences in the runtime environment. In contrast, the Budget RNN uses an adaptive controller that automatically handles these differences using runtime feedback.

E. Evaluation of Budget RNN System Design

The Budget RNN system uses a novel RNN with a runtime controller. We evaluate how these components contribute to the overall accuracy by considering two variants. The first, called the fixed system, uses a fixed model selection policy (§V-A)

Dataset	Wall-to-Wall Time (minutes)				Iterations (thousands)			
	RNN	*Phased*	*Skip*	*Budget*	*RNN*	*Phased*	*Skip*	*Budget*
EMG	40.3	557.1	464.9	176.9	81.4	803.6	724.8	288.9
FordA	9.4	90.7	92.5	35.5	45.6	293.4	320.9	137.7
Pavement	114.6	553.9	297.4	357.1	260.5	1139.9	664.8	552.6
Pedestrian	13.3	136.5	89.5	52.3	63.3	461.2	311.4	176.5
Pen Digits	24.0	233.1	109.6	92.0	188.8	1355.2	699.1	445.3
UCI HAR	124.0	1143.3	1039.0	346.1	251.7	1591.9	1559.8	512.1
Whale	40.0	393.0	500.1	119.1	112.1	734.0	988.3	272.2

Fig. 10: Wall-to-wall time and training iterations required to fit the neural networks on each dataset. The Phased and Skip RNN results account for training ten distinct RNNs. The Budget RNN values include the halting threshold optimization.

in place of the adaptive controller. The second variant uses a randomized selection policy. This policy creates weights based on how often the Budget RNN should stop at each level to fully utilize the budget. At runtime, the policy uses the weights to randomly select the number of levels. In this experiment, we call the full system the "adaptive" Budget RNN.

We evaluate these variants across the same datasets and budgets used in §V-B. Figure 9 shows the geometric mean accuracy for each dataset. The adaptive system outperforms the fixed and random variants on all datasets by a mean of 1.4 points and 3.1 points, respectively. These results yield two takeaways. First, data-dependent decisions are an integral part of the adaptive system's performance. The randomized variant's lower accuracy shows how fully utilizing the energy budget is insufficient; selecting when to conserve and when to exploit is key to obtaining accurate results. Second, the fixed variant alone displays an improvement over the standard RNN (compare "Fixed" in Figure 9 to "RNN" in Figure 5). This improvement shows the benefits of performing inference on non-contiguous subsequences. Adaptive behavior further exploits this feature to increase accuracy.

F. Training Time

One downside to the Phased and Skip RNN systems is their reliance on many distinct RNNs. We evaluate how this reliance impacts the training time of each system. We measure the training cost in two ways. The first is the wall-to-wall time required to train all models for each system. We obtain these times on a 24 core Intel Xeon Silver 4116 CPU. The second metric is the number of training iterations. This value is the number of gradient descent steps during training. The Budget RNN values also include the halting threshold optimization. As a conservative estimate, we count each threshold optimization step as equivalent to 10 gradient descent steps.

We measure the training cost on all seven datasets. Figure 10 displays the results. Compared to Skip RNNs, the Budget RNN system takes roughly $2.3\times$ less time and fewer iterations to train. A comparison with the Phased RNN systems yields even greater savings. While these costs could be lowered by reducing the number of models per system, such savings lead to lower accuracy. Only the standard RNN achieves a lower training cost than the Budget RNN. This property follows from the need to train only one neural network. As shown, the extra training cost of Budget RNNs leads to better accuracy values. We note that the Skip RNN system trains faster than the Budget RNN system on the pavement dataset. This result occurs due to early stopping during neural network training and a larger training set used to fit the halting thresholds. On this dataset, the benefits of the lower training time of Skip RNNs are offset by the system's lower accuracy.

Energy Budget	Skip RNN	Budget RNN
5.6J	0.722 (±0.040)	0.833 (±0.038)
7.2J	0.895 (±0.038)	0.915 (±0.022)

Fig. 11: Skip and Budget RNN results on the embedded device using the Pen Digits dataset.

G. Performance in the Hardware Environment

We supplement the results from the profiled environment with an evaluation in the embedded setting described in §V-A. We compare the accuracy of the Budget RNNs and Skip RNNs over two energy budgets on the Pen Digits dataset. We select Skip RNNs because they form the best-performing baseline on this task. As the device fetches inputs over a Bluetooth link, we use a Budget RNN system optimized on the Bluetooth energy profile. Each experiment uses the inference system to classify $M = 50$ sequences. We compare the resulting accuracy across four independent trials.

As shown in Figure 11, the Budget RNN displays better accuracy than the Skip RNNs. In the simulated environment, the Budget RNN system achieves an accuracy of 79.1% and 90.3% on the full dataset for the two respective budgets. On these same budgets, the Skip RNN system obtains an accuracy of 75.6% and 89.4%. Thus, in the embedded environment, the gap between Budget RNNs and Skip RNNs is slightly larger than what is observed in simulation. This discrepancy is likely a result of the limited sample size used in the hardware evaluation. Despite this difference, the results from the hardware device confirm the relative accuracy trends we observe in the profiled environment.

H. Overhead Analysis

Budget RNNs are designed for embedded applications where sensing composes a significant portion of energy consumption. In these settings, the computational overhead of Budget RNNs is not prohibitive. To understand this point further, we use a TI MSP430 FR5994 MCU [35] and the EnergyTrace Tool [62] to profile the energy consumption of collecting and processing a single measurement. We compare the energy consumption of standard and Budget RNNs in Figure 12. To handle a single element, Budget RNNs consume 0.5% (Bluetooth) and 2.7% (temperature) more energy than standard RNNs. This overhead is small enough to overcome and still deliver high accuracy; under the same constraints, Budget RNNs show better accuracy than the baseline RNNs (§V-B). Budget RNNs use more computation to determine a much better sampling strategy. The relatively high cost of sensing makes this additional computation worthwhile.

Sensor	Alone	+RNN	+Budget RNN
Bluetooth	29.63	29.97	30.13
Temperature	5.65	5.99	6.15

Fig. 12: Energy (mJ) required to handle a single sensor measurement. The standard and Budget RNN costs include the energy required for processing.

The computational overhead of Budget RNNs comes from two areas. First, Budget RNNs use merging, halting, and pooling layers to control execution and combine intermediate states. Second, the Budget RNN system uses a runtime controller to adapt to energy constraints. On average, profiling shows that standard RNNs consume 0.342mJ to process a single element. Budget RNNs use an average of 0.503mJ. A majority of this overhead comes from the added neural network layers. The runtime controller accounts for less than 3.5% of the CPU cycles associated with Budget RNN execution. As the controller performs all operations in RAM, we expect it to consume a smaller fraction of the overall energy.

VI. Related Work

A. Adapting Neural Network Inference

Many systems apply adaptation to improve neural network inference. Systems such as BranchyNet [63] and ALERT [64, 65] use early-exit points to control the inference cost through variable computation. Numerous systems use variable computation [66, 67, 68] and per-layer adaptivity [69, 70] to execute neural networks under explicit constraints. Neural network systems for multi-tenant inference also use adaptive behavior to improve efficiency [71, 72, 73, 74]. Other work adapts accuracy for energy or latency savings [75, 76, 77].

Other RNN designs adapt model execution. Adaptive Computation Time [19], Clockwork RNNs [20], and Variable Computation Time [78] alter the amount of computation at each RNN timestep. Shallow RNNs [23], Dilated RNNs [79], and Sliced RNNs [80] provide increased parallelism.

Budget RNNs are most similar to prior systems that use variable computation. Budget RNNs vary inference energy costs by changing both computation and data collection. In contrast with the above prior work that only reduces computation, Budget RNNs lower energy costs in sensor environments where acquiring data is expensive.

B. Adaptive Sampling in RNNs

Multiple RNNs leverage adaptive sampling. Phased LSTMs use a periodic gate to perform state updates [25]. LSTM-Jump uses a policy gradient to train an RNN to skip inputs [81]. Skip RNNs skip inputs using a binary gate [26]. These models jointly optimize sampling behavior and RNN parameters. Unlike these designs, Budget RNNs are multi-capacity models that can change their target sampling level at runtime.

EMI-RNNs classify time-series inputs where the true signal makes up a small part of the sequence [15]. This model performs early classification using thresholds on output probabilities. This design resembles a Budget RNN with a stride length of 1. These models, however, differ in their early-stopping criterion. Furthermore, EMI-RNNs use sliding windows during inference, so they still pay data acquisition costs.

C. Adaptive Sampling in Sensor Networks

Adaptive sampling is a common approach to conserve energy in sensor networks. Multiple systems use statistical models to control the sampling rate of individual sensors [9, 11, 82, 83, 84, 85]. Backcasting [10], ASAP [86], and BBQ [87] selectively activate a subset of nodes based on sensor correlations. Similar to these systems, Budget RNNs also use adaptive sampling to reduce energy while maximizing sensor performance. Unlike this prior work, our system focuses on meeting explicit energy budgets when executing RNNs.

D. Early Time Series Classification

Systems for early time series classification perform inference on subsequences. ECTS uses a nearest neighbor algorithm on sequence prefixes and halts classification when the prediction becomes reliable [88]. Mori et al. use stopping rules to balance accuracy and earliness [89]. Similar to Budget RNNs, these systems control inference costs by processing subsequences. Budget RNNs, however, perform inference under energy budgets and do not optimize for earliness.

Rußwurm et al. perform early classification on RNNs by randomly sampling trainable halting probabilities [41]. Budget RNNs also use halting signals to control model execution. Unlike this previous work, Budget RNNs use optimized halting thresholds to meet energy budgets without random behavior.

VII. Conclusion

This paper develops a novel RNN architecture—the Budget RNN—for in-sensor inference under energy budgets. The Budget RNN uses a leveled design in which each level processes an input subsequence. By controlling the number of executed levels, the Budget RNN can alter its energy consumption. This control is made possible through trainable halting signals. We design a runtime controller that uses these signals to perform inference under an energy constraint by dynamically adjusting the model's subsampling behavior. This design maintains sampling flexibility while also decoupling the sampling strategy from the RNN parameters. This loose-coupling allows the system to generalize to unseen budgets better than existing RNN solutions [25, 26], achieving a mean accuracy that is 3 points higher than that of standard RNNs. This improvement allows Budget RNNs to obtain an accuracy that is equivalent to existing RNNs even when operating under 20% smaller budgets.

Acknowledgements

We thank the reviewers, the artifact evaluators, and our anonymous shepherd for helping us improve our paper. This work is supported by NSF (grants CCF-2028427, CNS-1956180, CCF-1837120, CNS-1764039), ARO (grant W911NF1920321), and a DOE Early Career Award (grant DESC0014195 0003).

References

[1] L. M. Naylor and J. G. Kie, "Monitoring activity of Rocky Mountain elk using recording accelerometers," *Wildlife Society Bulletin*, vol. 32, no. 4, pp. 1108–1113, 2004.

[2] K. Martinez, R. Ong, and J. Hart, "Glacsweb: A sensor network for hostile environments," in *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.* IEEE, 2004, pp. 81–87.

[3] S. Tyler, D. Holland, V. Zagorodnov, A. Stern, C. Sladek, S. Kobs, S. White, F. Suárez, and J. Bryenton, "Using distributed temperature sensors to monitor an antarctic ice shelf and sub-ice-shelf cavity," *Journal of Glaciology*, vol. 59, no. 215, pp. 583–591, 2013.

[4] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in ZebraNet," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 227–238.

[5] Y. Yang and C. Wang, *Wireless rechargeable sensor networks*. Springer, 2015.

[6] R. Bagree, V. R. Jain, A. Kumar, and P. Ranjan, "Tigercense: Wireless image sensor network to monitor tiger movement," in *International Workshop on Real-world Wireless Sensor Networks*. Springer, 2010, pp. 13–24.

[7] S. Jang, H. Jo, S. Cho, K. Mechitov, J. A. Rice, S.-H. Sim, H.-J. Jung, C.-B. Yun, B. F. Spencer Jr, and G. Agha, "Structural health monitoring of a cable-stayed bridge using smart sensor technology: Deployment and evaluation," *Smart Structures and Systems*, vol. 6, no. 5_6, pp. 439–459, 2010.

[8] P. Sommer, B. Kusy, P. Valencia, R. Dungavell, and R. Jurdak, "Delay-tolerant networking for long-term animal tracking," *IEEE Internet Computing*, vol. 22, no. 1, pp. 62–72, 2018.

[9] A. Jain and E. Y. Chang, "Adaptive sampling for sensor networks," in *Proceeedings of the 1st international workshop on Data management for sensor networks: in conjunction with VLDB 2004*, 2004, pp. 10–16.

[10] R. Willett, A. Martin, and R. Nowak, "Backcasting: Adaptive sampling for sensor networks," in *Proceedings of the 3rd international symposium on Information processing in sensor networks*, 2004, pp. 124–133.

[11] J. Zhou and D. De Roure, "Floodnet: Coupling adaptive sampling with energy aware routing in a flood warning system," *Journal of Computer Science and Technology*, vol. 22, no. 1, pp. 121–130, 2007.

[12] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient machine learning in 2 KB RAM for the internet of things," in *International Conference on Machine Learning*, 2017, pp. 1935–1944.

[13] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[14] B. Denby and B. Lucia, "Orbital edge computing: Machine inference in space," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 59–62, 2019.

[15] D. Dennis, C. Pabbaraju, H. V. Simhadri, and P. Jain, "Multiple instance learning for efficient sequential data classification on resource-constrained devices," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 953–10 964.

[16] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 199–213.

[17] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 328–339.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[20] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, "A clockwork RNN," *arXiv preprint arXiv:1402.3511*, 2014.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[23] D. Dennis, D. A. E. Acar, V. Mandikal, V. S. Sadasivan, V. Saligrama, H. V. Simhadri, and P. Jain, "Shallow RNN: accurate time-series classification on resource constrained devices," in *Advances in Neural Information Processing Systems*, 2019, pp. 12 916–12 926.

[24] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, "FastGRNN: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network," *arXiv preprint arXiv:1901.02358*, 2019.

[25] D. Neil, M. Pfeiffer, and S.-C. Liu, "Phased LSTM: Accelerating recurrent network training for long or event-based sequences," in *Advances in neural information processing systems*, 2016, pp. 3882–3890.

[26] V. Campos, B. Jou, X. Giró-i Nieto, J. Torres, and S.-F. Chang, "Skip RNN: Learning to skip state updates in recurrent neural networks," *arXiv preprint arXiv:1708.06834*, 2017.

[27] L. He, L. Kong, Y. Gu, J. Pan, and T. Zhu, "Evaluating the on-demand mobile charging in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 14, no. 9, pp. 1861–1875, 2014.

[28] C. Lin, Y. Zhou, H. Dai, J. Deng, and G. Wu, "MPF: Prolonging network lifetime of wireless rechargeable sensor networks by mixing partial charge and full charge," in *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2018, pp. 1–9.

[29] S. Madden and M. J. Franklin, "Fjording the stream: An architecture for queries over streaming sensor data," in *Proceedings 18th International Conference on Data Engineering*. IEEE, 2002, pp. 555–566.

[30] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM SigKDD Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, 2011.

[31] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical review letters*, vol. 59, no. 19, p. 2229, 1987.

[32] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[33] J. Collins, J. Sohl-Dickstein, and D. Sussillo, "Capacity and trainability in recurrent neural networks," *arXiv preprint arXiv:1611.09913*, 2016.

[34] D. E. Rumelhart, B. Widrow, and M. A. Lehr, "The basic ideas in neural networks," *Communications of the ACM*, vol. 37, no. 3, pp. 87–93, 1994.

[35] "TI MSP430 FR5994 Datasheet," https://www.ti.com/lit/ds/symlink/msp430fr5994.pdf.

[36] "Atmel sam l21 datasheet," http://ww1.microchip.com/downloads/en/DeviceDoc/SAM_L21_Family_DataSheet_DS60001477C.pdf.

[37] C. Alippi, G. Anastasi, M. Di Francesco, and M. Roveri, "Energy management in wireless sensor networks with energy-hungry sensors," *IEEE Instrumentation & Measurement Magazine*, vol. 12, no. 2, pp. 16–23, 2009.

[38] A. Martins and R. Astudillo, "From softmax to sparsemax: A sparse model of attention and multi-label classification," in *International Conference on Machine Learning*, 2016, pp. 1614–1623.

[39] A. Graves, "Adaptive computation time for recurrent neural networks," *arXiv preprint arXiv:1603.08983*, 2016.

[40] J. Schmidhuber, "Self-delimiting neural networks," *arXiv preprint arXiv:1210.0118*, 2012.

[41] M. Rußwurm, S. Lefevre, N. Courty, R. Emonet, M. Körner, and R. Tavenard, "End-to-end learning for early classification of time series," *arXiv preprint arXiv:1901.10681*, 2019.

[42] T. T. Wu, K. Lange *et al.*, "Coordinate descent algorithms for lasso penalized regression," *The Annals of Applied Statistics*, vol. 2, no. 1, pp. 224–244, 2008.

[43] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in neural information processing systems*, 2001, pp. 402–408.

[44] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan *et al.*, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.

[45] D. E. Rivera, M. Morari, and S. Skogestad, "Internal model control: PID controller design," *Industrial & engineering chemistry process design and development*, vol. 25, no. 1, pp. 252–265, 1986.

[46] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 397–422, 2002.

[47] R. A. Fisher *et al.*, "A mathematical examination of the methods of determining the accuracy of an observation by the mean error, and by the mean square error." 1920.

[48] A. Ly, M. Marsman, J. Verhagen, R. P. Grasman, and E.-J. Wagenmakers, "A tutorial on Fisher information," *Journal of Mathematical Psychology*, vol. 80, pp. 40–55, 2017.

[49] T. M. Cox, T. Ragen, A. Read, E. Vos, R. W. Baird, K. Balcomb, J. Barlow, J. Caldwell, T. Cranford, and L. Crum, "Understanding the impacts of anthropogenic sound on beaked whales," Space and Naval Warfare Systems Center, San Diego, CA, Tech. Rep., 2006.

[50] S. Lobov, N. Krilova, I. Kastalskiy, V. Kazantsev, and V. A. Makarov, "Latent factors limiting the performance of sEMG-interfaces," *Sensors*, vol. 18, no. 4, p. 1122, 2018.

[51] A. Bagnall, L. Davis, J. Hills, and J. Lines, "Transformation based ensembles for time series classification," in *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 2012, pp. 307–318.

[52] V. M. Souza, "Asphalt pavement classification using smartphone accelerometer and complexity invariant distance," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 198–211, 2018.

[53] "Melbourne pedestrian counter," http://www.pedestrian.melbourne.vic.gov.au/.

[54] "Melbourne pedestrian dataset," http://www.timeseriesclassification.com/description.php?Dataset=MelbournePedestrian.

[55] F. Alimoglu and E. Alpaydin, "Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition," in *Proceedings of the fifth turkish artificial intelligence and artificial neural networks symposium*. Citeseer, 1996.

[56] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*. Springer, 2012, pp. 216–223.

[57] "Right whale calls dataset," http://www.timeseriesclassification.com/description.php?Dataset=RightWhaleCalls.

[58] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, vol. 30, no. 1, 2013, p. 3.

[59] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[60] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[61] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[62] "TI EnergyTrace Tool," https://www.ti.com/lit/an/slaa603/slaa603.pdf?ts=1603499847200.

[63] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[64] C. Wan, H. Hoffmann, S. Lu, and M. Maire, "Orthogonalized SGD and nested architectures for anytime neural networks," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 9807–9817. [Online]. Available: http://proceedings.mlr.press/v119/wan20a.html

[65] C. Wan, M. H. Santriaji, E. Rogers, H. Hoffmann, M. Maire, and S. Lu, "ALERT: accurate learning for energy and timeliness," in *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*, A. Gavrilovska and E. Zadok, Eds. USENIX Association, 2020, pp. 353–369. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/wan

[66] M.-H. Cheng, Q. Sun, and C.-H. Tu, "An adaptive computation framework of distributed deep learning models for internet-of-things applications," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 85–91.

[67] S. Lee and S. Nirjon, "SubFlow: A dynamic induced-subgraph strategy toward real-time DNN inference and training," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 15–29.

[68] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. Abdelzaher, "Scheduling real-time deep learning services as imprecise computations," in *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2020, pp. 1–10.

[69] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 67–79.

[70] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 107–118.

[71] I. Baek, M. Harding, A. Kanda, K. R. Choi, S. Samii, and R. R. Rajkumar, "CARSS: Client-aware resource sharing and scheduling for heterogeneous applications," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 324–335.

[72] D. Casini, A. Biondi, and G. Buttazzo, "Timing isolation and improved scheduling of deep neural networks for real-time systems," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1760–1777, 2020.

[73] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 115–127.

[74] S. Bateni and C. Liu, "NeuOS: A Latency-Predictable Multi-Dimensional Optimization Framework for DNN-driven Autonomous Systems," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 371–385.

[75] H. Hoffmann, "JouleGuard: Energy guarantees for approximate applications," in *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015*, E. L. Miller and S. Hand, Eds. ACM, 2015, pp. 198–214.

[76] A. Farrell and H. Hoffmann, "MEANTIME: Achieving both minimal energy and timeliness with approximate computing," in *2016 USENIX Annual Technical Conference, USENIX ATC 2016*, A. Gulati and H. Weatherspoon, Eds. USENIX Association, 2016, pp. 421–435.

[77] H. Hoffmann, "CoAdapt: Predictable behavior for accuracy-aware applications running on power-aware systems," in *26th Euromicro Conference on Real-Time Systems, ECRTS 2014*. IEEE Computer Society, 2014, pp. 223–232.

[78] Y. Jernite, E. Grave, A. Joulin, and T. Mikolov, "Variable computation in recurrent neural networks," *arXiv preprint arXiv:1611.06188*, 2016.

[79] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson, and T. S. Huang, "Dilated recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 77–87.

[80] Z. Yu and G. Liu, "Sliced recurrent neural networks," *arXiv preprint arXiv:1807.02291*, 2018.

[81] A. W. Yu, H. Lee, and Q. V. Le, "Learning to skim text," *arXiv preprint arXiv:1704.06877*, 2017.

[82] C. Alippi, G. Anastasi, C. Galperti, F. Mancini, and M. Roveri, "Adaptive sampling for energy conservation in wireless sensor networks for snow monitoring applications," in *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*. IEEE, 2007, pp. 1–6.

[83] C. Alippi, G. Anastasi, M. Di Francesco, and M. Roveri, "An adaptive sampling algorithm for effective energy management in wireless sensor networks with energy-hungry sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 2, pp. 335–344, 2009.

[84] S. Chatterjea and P. Havinga, "An adaptive and autonomous sensor sampling frequency control scheme for energy-efficient data acquisition in wireless sensor networks," in *International Conference on Distributed Computing in Sensor Systems*. Springer, 2008, pp. 60–78.

[85] Y. W. Law, S. Chatterjea, J. Jin, T. Hanselmann, and M. Palaniswami, "Energy-efficient data acquisition by adaptive sampling for wireless sensor networks," in *Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly*, 2009, pp. 1146–1151.

[86] B. Gedik, L. Liu, and S. Y. Philip, "ASAP: An adaptive sampling approach to data collection in sensor networks," *IEEE Transactions on Parallel and distributed systems*, vol. 18, no. 12, pp. 1766–1783, 2007.

[87] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, 2004, pp. 588–599.

[88] Z. Xing, J. Pei, and S. Y. Philip, "Early classification on time series," *Knowledge and information systems*, vol. 31, no. 1, pp. 105–127, 2012.

[89] U. Mori, A. Mendiburu, S. Dasgupta, and J. A. Lozano, "Early classification of time series by simultaneously optimizing the accuracy and earliness," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 10, pp. 4569–4578, 2017.