# Intermittent-Aware Neural Architecture Search

HASHAN ROSHANTHA MENDIS, Academia Sinica, Taiwan
CHIH-KAI KANG, Academia Sinica and National Taiwan University, Taiwan
PI-CHENG HSIU, Academia Sinica, National Taiwan University and National Chi Nan University, Taiwan

The increasing paradigm shift towards *intermittent computing* has made it possible to intermittently execute *deep neural network* (DNN) inference on edge devices powered by ambient energy. Recently, *neural architecture search* (NAS) techniques have achieved great success in automatically finding DNNs with high accuracy and low inference latency on the deployed hardware. We make a key observation, where NAS attempts to improve inference latency by primarily maximizing data reuse, but the derived solutions when deployed on intermittently-powered systems may be inefficient, such that the inference may not satisfy an end-to-end latency requirement and, more seriously, they may be unsafe given an insufficient energy budget.

This work proposes iNAS, which introduces intermittent execution behavior into NAS to find accurate network architectures with corresponding execution designs, which can safely and efficiently execute under intermittent power. An intermittent-aware execution design explorer is presented, which finds the right balance between data reuse and the costs related to intermittent inference, and incorporates a preservation design search space into NAS, while ensuring the power-cycle energy budget is not exceeded. To assess an intermittent execution design, an intermittent-aware abstract performance model is presented, which formulates the key costs related to progress preservation and recovery during intermittent inference. We implement iNAS on top of an existing NAS framework and evaluate their respective solutions found for various datasets, energy budgets and latency requirements, on a Texas Instruments device. Compared to those NAS solutions that can safely complete the inference, the iNAS solutions reduce the intermittent inference latency by 60% on average while achieving comparable accuracy, with an average 7% increase in search overhead.

CCS Concepts: • **Computer systems organization → Embedded software**; • **Computing methodologies → Neural networks**.

Additional Key Words and Phrases: Deep neural networks, neural architecture search, design space exploration, intermittent systems, energy harvesting, edge computing

**ACM Reference Format:**
Hashan Roshantha Mendis, Chih-Kai Kang, and Pi-Cheng Hsiu. 2021. Intermittent-Aware Neural Architecture Search. *ACM Trans. Embedd. Comput. Syst.* 20, 5s, Article 64 (September 2021), 27 pages. https://doi.org/10.1145/3476995

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2021.
This work was supported in part by the Ministry of Science and Technology, Taiwan, under grant MOST 110-2222-E-001-003-MY3.
Authors' addresses: Hashan Roshantha Mendis, Research Center for Information Technology Innovation (CITI), Academia Sinica, Taiwan, rosh.mendis@citi.sinica.edu.tw; Chih-Kai Kang, Research Center for Information Technology Innovation (CITI), Academia Sinica and Graduate Institute of Electrical Engineering, National Taiwan University, Taiwan, ckkang@arbor.ee.ntu.edu.tw; Pi-Cheng Hsiu (corresponding author), Research Center for Information Technology Innovation (CITI), Academia Sinica, College of Electrical Engineering and Computer Science, National Taiwan University, and Department of Computer Science and Information Engineering, National Chi Nan University, Taiwan, pchsiu@citi.sinica.edu.tw.


1539-9087/2021/9-ART64 $15.00
https://doi.org/10.1145/3476995

## 1 INTRODUCTION

Energy harvesting has emerged as a sustainable and cost-effective solution for modern edge devices by relying on ambient energy rather than battery-based power sources [48]. Applications on these devices suffer from frequent power failure and thus execute *intermittently*, when energy is available [13, 24]. These devices consume orders-of magnitude more energy to communicate with a remote server than local computation or sensing [19]. Therefore, the need for responsive edge applications with low communication bandwidth consumption has motivated the development of on-device intelligence, focusing research efforts on how to execute *deep neural network (DNN) inference* on *intermittent systems*, across power cycles [19, 36]. DNN models have an intractable design space which makes it laborious to manually design models even with expert knowledge, and hence *neural architecture search (NAS)* techniques were developed to automatically find highly accurate neural networks that can efficiently execute on the deployed systems [34, 70]. With the widespread automated development of deep neural networks and rising demand for deployment on battery-less edge devices, *intermittent-aware neural architecture search* is emerging as a crucial problem to solve.

DNN inference under intermittent power has recently been enabled, by accumulatively executing inference across power cycles [19, 36]. As ambient power is typically unstable and too weak for continuous execution [47], even extremely power-efficient hardware [10, 21], may take multiple power-cycles to complete a full inference. Therefore, intermittent inference execution is necessary on lightweight energy-harvesting devices, and in most cases it cannot be simply avoided by using power efficient hardware. *Intermittent execution behavior* differs significantly from continuous execution behavior. Inference must safely execute within an *energy budget* in each power cycle and intermittent inference approaches ensure the overhead of *progress preservation* during inference do not impede performance and guarantee correct *progress recovery* to resume inference upon power resumption. Progress preservation includes backing up inference progress indicators along with inference computation outputs from volatile memory (VM) to non-volatile memory (NVM). Subsequently, during progress recovery, the progress indicators are fetched from NVM to VM, following which the input data fetched into VM in the previous power cycle but lost due to power loss are re-fetched, to resume the interrupted inference process. Intermittent inference approaches may differ in terms of what data they preserve during inference and how much progress is re-executed upon power resumption [19, 36], but all approaches simply take a given pre-trained DNN model with a *fixed* architecture and execute it intermittently on edge devices.

Neural architecture search is a design-time process performed at the server-side that explores the neural architecture space to find DNNs with high accuracy for a particular application/dataset [70]. The derived DNNs are then deployed (e.g., onto edge devices) to perform inference, but there is no guarantee that they will meet the required performance after being implemented on the target hardware platform. To overcome this challenge, recent research has proposed *hardware-aware NAS* (referred to as HW-NAS) approaches [22, 32, 34], which co-explore both the *neural architecture space* and *execution design space*, to find DNNs with high accuracy and associated *execution designs* that do not violate a target inference latency requirement. However, all NAS approaches assume the deployed system executes under continuous power, i.e., without power failure.

This work is motivated by our key findings related to the unsuitability of existing NAS approaches (including hardware-aware variants) for intermittently-powered inference systems, that cannot be resolved via straightforward extensions. We observe that NAS primarily seeks to *maximize data reuse* without being aware of intermittent execution behavior, and therefore an optimal neural network and associated execution design found by NAS may be *inefficient* and more seriously, it may even be *unsafe*, when deployed on an intermittent inference system. Maximizing data

reuse unavoidably leads to large, energy expensive inference execution designs that may fail to safely execute within a power cycle, resulting in repeated re-execution of an inference, without termination. In addition, solely maximizing data reuse without making appropriate trade-offs between data reuse and additional costs such as data re-fetching cost, may lead to an increase in the progress recovery cost. Moreover, enforcing hardware specification constraints without considering crucial design spaces related to intermittent inference, such as batch preservation (i.e., multiple computation outputs preserved together to reduce preservation overhead), leads to severe energy budget underutilization. As a consequence, these factors may result in inefficient execution designs that violate the end-to-end inference latency requirement. Therefore, as a general principle, to generate and deploy neural networks on intermittent systems, an intermittent-aware neural architecture search should find the right balance between data reuse and the costs related to progress preservation and recovery, while ensuring the power-cycle energy budget is not exceeded.

In this work, we present *iNAS (intermittent-aware neural architecture search)*, the first framework which follows the general principle presented above, to find accurate neural networks that can *safely* and *efficiently* execute under intermittent power. Introducing intermittent execution behavior into NAS raises two key design challenges. The first challenge is how to define a *feasible solution space* that captures intermittent execution behavior combined with existing NAS execution design. To address this challenge, we present an *intermittent-aware design space explorer* that introduces a new preservation design space unique to intermittent inference, which is parameterized and co-searched with the conventional execution design space, while enforcing an energy budget constraint at the power-cycle level, hardware constraints adapted to consider the joint design space, and an inference latency constraint specified by the application. The second challenge is how to analytically *formulate the performance* of an intermittent execution design that accurately captures all necessary costs. To address this challenge, we present an *intermittent-aware abstract performance model*, which formulates the intermittent inference costs related to progress preservation, recovery and computation, using the tightly coupled parameters of the joint design space, and subsequently, these costs considered across the multiple power-cycle execution of each network layer are combined to derive the end-to-end intermittent inference latency.

We implemented iNAS on top of an existing HW-NAS framework [34], although the proposed intermittent-aware co-exploration is sufficiently decoupled to allow easy integration with most existing NAS frameworks. The solutions derived by iNAS and the aforementioned HW-NAS were deployed and evaluated on a Texas Instruments MSP430FR5994 microcontroller unit (MCU), with a low energy accelerator (LEA), 8KB SRAM (VM) and external 1MB FRAM (NVM). Experiments were conducted under intermittent power with different capacitor (energy buffer) sizes and latency requirements. Three datasets, representative of those typically generated by tiny machine learning applications [6] were used for evaluation, with each dataset using a different model architecture type with varied levels of search space complexity. Compared to the HW-NAS solutions that can safely complete the inference, the iNAS solutions show a 16% to 87% reduction in intermittent inference latency while maintaining a similar solution accuracy, at a cost of 0.1% to 17% additional search overhead. The latency improvements are more obvious when it is particularly difficult for HW-NAS to find feasible solutions, such as under a small capacitor, where the HW-NAS solutions do not make forward progress, as well as under a high latency requirement, where more complex network architectures are found. We also conduct secondary experiments which enabled us to provide useful system design guidelines for configuring new intermittent inference platforms. Specifically, we show how the relationship between the model architecture, VM size and capacitor size has a significant impact on the end-to-end inference latency. This demonstrates the practicality of iNAS as a tool to provide design considerations related to the target application requirements, early in the design process.

We summarize the contributions of this work as follows:

- We introduce the novel problem of intermittent-aware neural architecture search and present our key findings on the unsuitability of HW-NAS for intermittent inference. Subsequently, we provide a general principle that should be followed to find safe and efficient DNNs for intermittent systems.
- We present two key challenges that arise when realizing an intermittent-aware NAS, namely, defining the feasible solution space and formulating the performance of an intermittent execution design. To address these challenges, we propose an intermittent-aware design space explorer and intermittent-aware abstract performance model.
- We make the developed iNAS framework publicly available [52], allowing AI practitioners to automate the design and deployment of DNNs on energy harvesting edge devices. We also demonstrate the practicality of iNAS as a tool to facilitate application specific, early platform configuration decisions.

The remainder of this paper is organized as follows. Section 2 provides background information and Section 3 explains the motivation for this work. Section 4 presents our iNAS design and Section 5 gives implementation details. Experimental results are reported in Section 6 and Section 7 discusses the limitations in iNAS and future extensions. Section 8 provides a discussion on related work and Section 9 presents some concluding remarks.

## 2 BACKGROUND

### 2.1 Deep Neural Network Design Space

DNNs consist of multiple sequential layers such as convolutional (CONV), pooling (POOL) and fully connected (FC) layers, where one layer's output is the subsequent layer's input. A CONV layer, for example, takes input feature maps (IFMs) with $N$ channels, each with a size $H \times W$ and convolves them with $M$ weight kernels of size $K \times K \times N$, to produce $M$ output feature maps (OFMs), each with a size $R \times C$ [58]. The architecture of each layer directly affects the *accuracy* and inference *latency* of a DNN model. DNN inference consists of loop-heavy and data intensive computations, thus requiring execution design considerations such as *data reuse* to allow deployment on resource constrained edge devices. Data reuse schemes primarily consider the loop tile size and loop order.

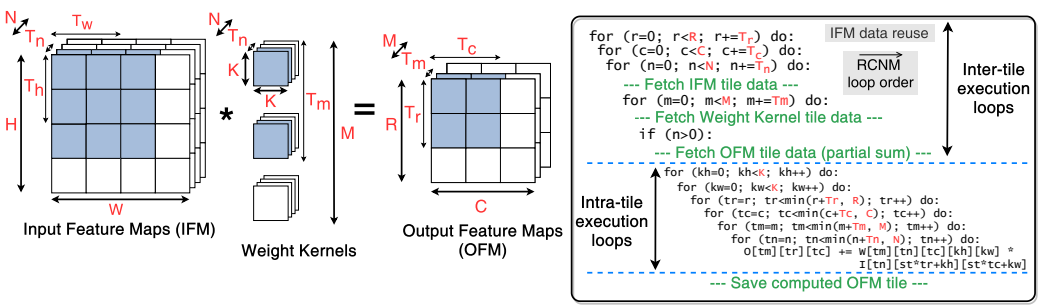


Fig. 1. Tiled convolution execution design

Tiled DNN computation is a common strategy used to overcome limited local memory. Due to their large size, a layer's IFMs, weights and OFMs are stored in NVM, and are logically partitioned into *tiles* as shown in Figure 1, according to the *tile size* parameters $< T_r, T_c, T_n, T_m >$. Although NVM

is typically larger than VM, for performance and energy concerns, tiles are generally processed in VM. A tile can be computed either using a hardware accelerator or CPU. To compute an OFM tile, *tile input data*, consisting of an *IFM tile*, *weight kernel tile* and *OFM tile* are first fetched from NVM to VM. An IFM tile is of size $T_h \times T_w \times T_n$ (where, $T_h = st \times T_r + K - st$, $T_w = st \times T_c + K - st$), a weight kernel tile has $T_m$ kernels, each with size $K \times K \times T_n$ and stride $st$, and an OFM tile is of size $T_r \times T_c \times T_m$ consisting of previously computed partial sums. The *intra-tile* execution loops (Figure 1 - bottom) compute the partial sums of the current tile and accumulate the result onto the previously computed partial sums. Energy harvesting, intermittent inference systems typically favor ultra-low power off-the-shelf MCUs, with lightweight hardware accelerators that can only support simple vector math operations [17, 26, 28, 37]. However, depending on the resources available on the hardware accelerator, one or more of the intra-tile loops can be unrolled and executed as a single accelerator operation, performing many multiply-accumulate (MAC) computations in parallel [34, 56, 68]. Lastly, the *tile output data* (i.e., computed OFM tile) are written back to an *OFM buffer* in NVM. This process is repeated for all tiles in a layer, where a layer has $\lceil \frac{R}{T_r} \rceil \lceil \frac{C}{T_c} \rceil \lceil \frac{N}{T_n} \rceil \lceil \frac{M}{T_m} \rceil$ number of tiles.

The loop order is typically selected to *maximize data reuse*, so that the amount of data movement between NVM and VM when the system is processing successive tiles can be minimized, thus reducing the inference latency [40, 49]. A loop order can be largely classified as being an *IFM*, *weight* or *OFM reuse order*, where the corresponding IFM, weight or OFM tile data fetched at the start of a particular inter-tile loop are reused in all iterations of that loop. For example, Figure 1 shows the IFM reuse order (i.e., $\overrightarrow{RCNM}$ inter-tile loop order), where the IFM tile data are fetched only once at the start of the $m$ inter-tile loop and *reused* in all $\lceil \frac{M}{T_m} \rceil$ iterations, while the weight and OFM tile data are fetched in each iteration. Similarly, weight and OFM reuse can be exploited by respectively using the $\overrightarrow{MNRC}$ loop order across the $r$ and $c$ inter-tile loops and the $\overrightarrow{RCMN}$ loop order across the $n$ inter-tile loop. The loop order that maximizes data reuse for a DNN layer depends heavily on the layer type and its dimensions.

### 2.2 Intermittent Deep Inference

Intermittent systems harvest and accumulate energy into an *energy buffer* (e.g., a capacitor) [13, 31]. The system is powered ON when the buffered energy reaches a preset threshold ($V_{on}$), and subsequently powered OFF when the energy buffer is depleted or reaches a low threshold ($V_{off}$). Hence, the buffered energy defines the available *energy budget* (denoted $E_{av}$) in a power cycle. The power ON duration is largely determined by both the energy budget and the energy consumption of the system, and the power OFF duration is related to the energy buffer recharge time, which depends on the energy buffer size and strength of the power supply.

Intermittent execution performs *progress preservation* during execution and *progress recovery* upon power resumption to overcome execution state and data loss due to power failure. Progress preservation involves backing up *progress indicators* along with computation outputs in NVM. Progress recovery involves system reboot (i.e., MCU cold-start and peripheral initialization), restoring the executing state using the preserved progress indicators and re-fetching the input data for computation, allowing the interrupted execution to be resumed. Power loss halts execution, so interrupted atomic (i.e., power uninterruptible) operations are *re-executed* upon power resumption, wasting energy and increasing the execution latency. For example, task-based intermittent execution models [13, 50] partition an application into atomic tasks, where the energy budget must satisfy the *energy consumption* of each task and progress preservation is performed after task completion. Upon power resumption, the interrupted task is recovered and re-executed. Alternatively, expensive re-execution can be avoided when the system is *proactively shut down* after completing

an atomic operation (e.g., a task) and before the system completely loses power [31, 35]. Here, as the energy buffer is not completely depleted, the *recharge time* required to charge the energy buffer back to the system power ON voltage threshold is reduced. However, the atomic operations need to be suitably sized to execute safely within the energy budget while avoiding underutilization.

Prior work in intermittent DNN inference has adopted a task-based execution model, where one or more output features or a complete layer tile (Section 2.1) are computed within a task [19]. Here, the *inter-tile loop indices* represent progress indicators, which are preserved along with *tile output data* as part of progress preservation during inference. Upon power resumption, the system is rebooted, the inter-tile loop indices are fetched and restored from NVM and the *tile input data* are *re-fetched* to resume the inference process from the interrupted inter-tile loop iteration. *Batch preservation* is used to reduce the progress preservation overhead [36], by retaining a batch of $S$ inference computation outputs (e.g., a batch of partial sums, output features or tile outputs) in VM and preserving the $S$ inference outputs together. The batch size $S$, needs to be appropriately set to balance preservation overhead and progress loss due to power failure, as a small $S$ may incur unnecessary preservation overhead under a relatively large energy budget and, contrarily, a large $S$ may not be able to make progress if the energy budget is too small. Moreover, additional VM may be required to retain $S$ computation outputs.

### 2.3 Neural Architecture Search

NAS finds the DNN architecture which maximizes the accuracy for a particular dataset [70]. The DNN architecture with the highest accuracy found by the NAS is then directly deployed on a hardware platform without any guarantee to its timing behavior, which may be problematic if inference latency is a critical factor. To overcome this issue, *hardware-aware NAS* (HW-NAS) frameworks *co-explore* both the neural architecture and execution design spaces [22, 32, 34] to find DNN architectures with high accuracy as well as feasible execution designs that do not violate a target *latency requirement* ($Lat_{req}$).

A typical NAS framework consists of a *NAS controller* and a *DNN trainer*. The NAS controller searches the *neural architecture space* and is commonly implemented as a recurrent neural network and updated using reinforcement learning [70]. It iteratively generates *child networks* with predicted DNN hyper-parameters (e.g., kernel size $K$ and number of filters $M$ in a CONV layer), improving the network architecture in each iteration, based on a reward signal. Conventionally, the reward is solely *accuracy*, obtained by an expensive training process carried out by the DNN trainer.

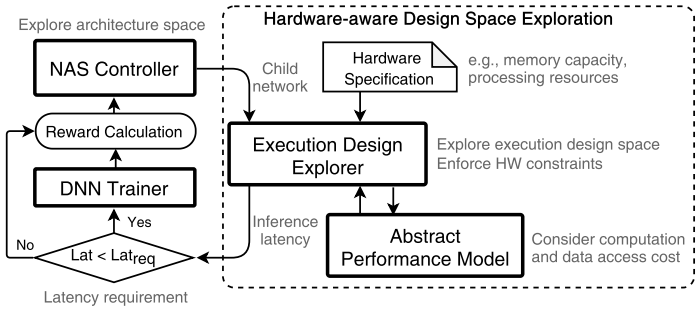

Fig. 2. A typical hardware-aware NAS framework

HW-NAS introduces an *execution design explorer*, an *abstract performance model* and a *hardware specification*, to a typical NAS framework, as shown in Figure 2. The execution design explorer takes a generated child network and the hardware specification (e.g., available VM and processing resources) as inputs to find the combination of *execution design parameters* (e.g., tile size and loop

order) that maximizes data reuse in order to minimize the inference latency ($Lat$) on the target platform [56, 68]. We refer to a specific combination of execution design parameters as a *candidate design*. Hardware specification constraints ensure a candidate design can be feasibly implemented on the target platform (e.g., tile memory requirement does not exceed the available VM capacity). An *abstract performance model* (e.g., [68]) is used to analytically estimate the inference latency of a candidate design without requiring real deployment on the target platform, considering the computation and data access costs related to inference execution. Note that different candidate designs impact inference latency, while accuracy is unaffected. A candidate design is considered feasible if it satisfies both the hardware specification and the latency requirement constraints (i.e., $Lat < Lat_{req}$). Once the feasible candidate designs have been found, the child network is trained on the target dataset. The achieved accuracy is combined with the lowest inference latency found by any of the feasible candidate designs and returned as a reward to the NAS controller. The child network is not trained if no feasible candidate designs can be found, and instead a negative reward is returned to the NAS controller to guide the search away from such infeasible networks.

## 3 NAS FOR INTERMITTENT SYSTEMS: OBSERVATIONS AND LIMITATIONS

Fundamentally, NAS assumes DNNs are deployed on continuously-powered systems. As such, it does not consider the characteristics of intermittent inference execution, such as execution across multiple power cycles, an energy budget in each power cycle and the progress preservation and recovery costs incurred in each power cycle. Therefore, we observe that an optimal neural network and inference execution design found by any NAS framework may be *inefficient* and, more seriously, *unsafe* when deployed on intermittently-powered systems.

Consider an example scenario where the NAS controller generates a child network consisting of a single CONV layer with an IFM of size $H = W = 16$, $N = 16$ input channels and $M = 32$ weight kernels each of size $K = 5$, resulting in an OFM of size $R = C = 12$ and $M = 32$ output channels. Assume the hardware specification includes VM with 4 KB capacity and a computational accelerator that performs $T_n$ MACs per operation. Accordingly, the combination of execution parameters (i.e., candidate design) that minimizes the inference latency (as discussed in Section 2.3) consists of the tile size parameters $T_r = 4$, $T_c = 6$, $T_m = 1$, $T_n = 16$ and the $IFM$ reuse loop order. The rationale behind this parameter selection is that NAS attempts to *maximize data reuse* to improve inference latency. As the CONV layer in the child network has a relatively large IFM and output channel size, large $T_r$, $T_c$ and $T_n$ tile parameters along with the IFM reuse order can maximize IFM tile data reuse. In this example, we refer to this specific candidate design found by a HW-NAS as the *baseline design*.

We consider the performance of the baseline design under intermittent power, using a task-based intermittent inference approach combined with a proactive shutdown strategy as discussed in Section 2.2. A task fetches the tile input data and computes the tile. After task completion, progress preservation is performed and the system is proactively shut down. Upon power resumption, progress recovery is performed and the inference is resumed from the subsequent task. Considering the intermittent execution costs related to tile processing, progress preservation and recovery, the energy consumption per power cycle (denoted $E_{pc}$) of the baseline can be estimated as $E_{pc} = 0.3$ mJ.

The candidate design derived by NAS may be *unsafe* under intermittent power; that is, the inference execution may fail if the energy consumed within a power cycle exceeds the energy budget. This is primarily because NAS strives to maximize data reuse, resulting in large tile sizes with higher energy cost, yet NAS does not consider the *energy budget* of a power cycle. Figure 3(a) provides an illustrative example where the relatively small energy budget is insufficient for the tile processing energy cost (i.e., $E_{av} = 0.2$ mJ, where $E_{av} < E_{pc}$), so the baseline design cannot safely make forward progress and will repeatedly re-execute without successful completion.

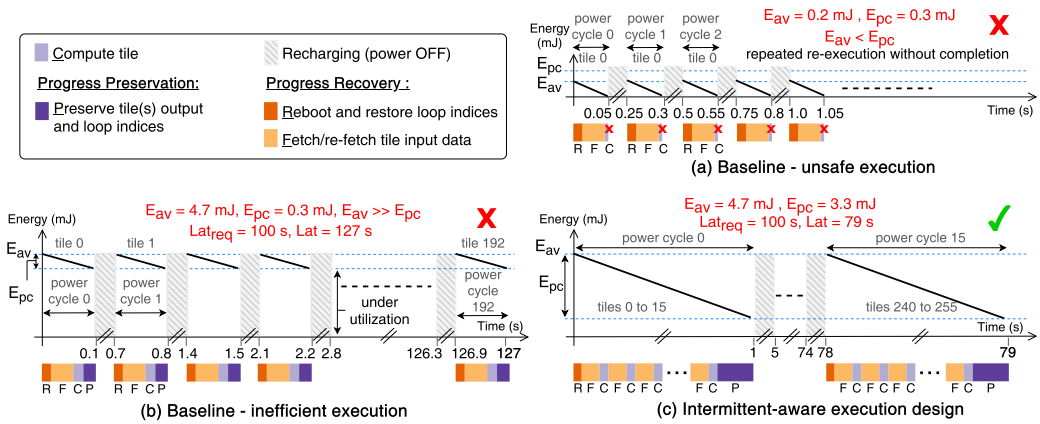


Fig. 3. A motivating example

Although forward progress is possible under a sufficiently large energy budget, the derived candidate design may be *inefficient*; that is, the end-to-end inference latency requirement may not be satisfied (i.e., $Lat > Lat_{req}$). This inefficiency is primarily because NAS does not make appropriate trade-offs between data reuse and intermittent execution costs (such as those incurred by progress preservation and recovery), and also because NAS does not consider the characteristics of intermittent inference (such as batch preservation) and related key design spaces. Solely maximizing data reuse may be disadvantageous because VM suffers from data loss upon power failure, which results in more data being re-fetched during progress recovery in the next power cycle, to process the subsequent tiles (Section 2.2). Similarly, solely considering hardware constraints such as the VM capacity size, without considering characteristics such as batch preservation (i.e., assuming a fixed preservation batch size of $S=1$), leads to intermittent execution designs that significantly underutilize the power-cycle energy budget, thus requiring a large number of power cycles to complete inference. Accordingly, Figure 3(b) shows an illustrative example where the baseline design severely underutilizes the energy budget, requiring 192 power cycles to complete the inference, thus resulting in a high $Lat$[1] of 127 s, which exceeds the $Lat_{req}$ of 100 s.

In contrast, Figure 3(c) shows an appropriate candidate design where the parameters were chosen considering *intermittent execution behavior* (i.e., tile size $T_r=3, T_c=6, T_m=1, T_n=16, S=16$ and the $IFM$ reuse order). To ensure safety and efficiency, this intermittent-aware candidate design balances data reuse and the additional costs related to progress preservation and recovery, and also considers batch preservation with a batch size of $S=16$, while ensuring the power-cycle energy consumption is within the energy budget. The relatively smaller tile size enables the system to accommodate the additional VM space required to retain the computed output of $S$ tiles. Here, batch preservation not only allows IFM tile data to be reused across the tiles processed in the same power cycle but also improves the energy budget utilization, thus reducing the number of power cycles required for intermittent inference. The intermittent-aware candidate design completes in 16 power cycles and has an end-to-end inference latency of 79 s, which satisfies the target latency requirement and significantly outperforms the baseline.

[1]The end-to-end inference latency includes the recharge time, assuming a constant but weak power supply. The recharge time may be higher if there is no energy to harvest.

# 4 INAS: INTERMITTENT-AWARE NEURAL ARCHITECTURE SEARCH

## 4.1 Design Rationale and Challenges

Motivated by the observations and limitations discussed in Section 3, we propose to introduce *intermittent execution behavior* into NAS. We enable NAS to consider the *energy budget* as well as the *additional costs* related to progress preservation and recovery, allowing NAS to find an accurate neural network which can safely and efficiently execute under intermittent power, without violating the inference latency requirement. However, introducing such a concept to NAS raises two key challenges.

The first challenge is how to define a *feasible solution space* which considers intermittent execution behavior. This is particularly difficult as preservation design parameters and energy budget related to intermittent execution are tightly coupled with conventional execution design parameters as well as hardware specification and inference latency constraints. To address this challenge, we present a novel *intermittent-aware execution design explorer* that introduces a preservation design space and an energy budget constraint to NAS. The preservation design space includes the preservation batch size parameter to specify the number of inference computation outputs preserved together, and it is unified and jointly explored with the execution design space. The energy budget constraint is combined with the hardware and latency constraints and enforced together during exploration.

The second challenge is how to analytically *formulate the performance* of an intermittent execution design solution. This is non-trivial as the additional key costs related to progress preservation and recovery have to be accurately captured. To address this challenge, we present an *intermittent-aware abstract performance model*. We formulate the power-cycle energy consumption and latency, considering the additional costs of preserving the progress indicators and inference computations as part of progress preservation, along with the costs of rebooting, fetching progress indicators and data re-fetching as part of progress recovery. These costs are based on the parameters in the unified preservation and execution design spaces, and considering these costs allows iNAS to find an intermittent execution design that can appropriately balance data reuse and the cost of data re-fetching. The formulated power-cycle energy and latency are used to derive the end-to-end inference latency across power cycles.

## 4.2 iNAS Overview

We design an intermittent-aware neural architecture search (iNAS) framework with the following problem definition. Given a specific dataset, a target intermittent system and an inference latency requirement ($Lat_{req}$), the objective is to generate a neural network, such that its end-to-end inference latency ($Lat$) on the given intermittent system is less than $Lat_{req}$, while achieving the maximum accuracy ($Acc$) on the given machine learning dataset. iNAS co-explores the network architecture, execution design and preservation design search spaces to find the most accurate network model and the most efficient intermittent execution design. iNAS employs analytical models within the design space exploration to derive the power-cycle energy consumption $E_{pc}$ and end-to-end inference latency $Lat$ to ensure safe and efficient inference execution under intermittent power.

As shown in Figure 4, iNAS consists of two key components: (1) the intermittent-aware execution design explorer (referred to as *iNAS-Exp*) and (2) the intermittent-aware abstract performance model (referred to as *iNAS-PMod*). They work alongside conventional NAS components, such as the NAS controller and the DNN trainer (Section 2.3). The NAS controller is based on [34], which uses a recurrent neural network to search the network architecture space and conducts reinforcement learning to update the controller. In each iteration, the NAS controller generates a

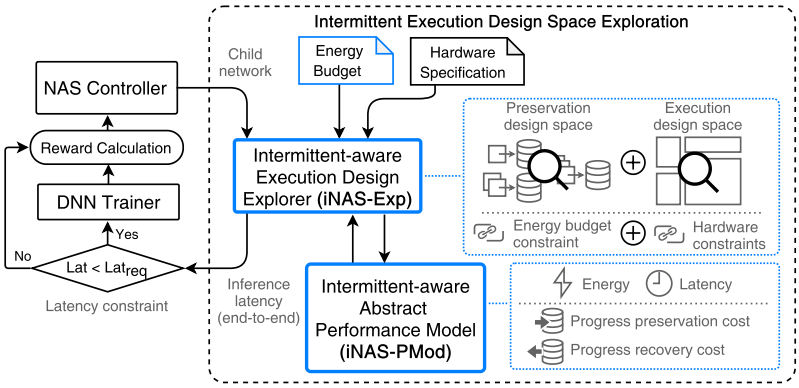


Fig. 4. Proposed Intermittent-aware NAS framework

child network, parameterized by the kernel size ($K$) and number of weight kernels ($M$) per layer, both of which crucially affect model accuracy and inference latency [23]. Hence, the size of the architecture space is $(|K|\times|M|)^L$, where $|K|$ and $|M|$ indicate the sizes of the discrete sets of kernel sizes and number of weight kernels per layer to search from, and $L$ is the number of network layers. Although we use this architecture search space to implement and show the benefit of introducing intermittent execution behavior into NAS, more sophisticated NAS controllers with a larger, parameter-rich architecture space can be easily integrated into iNAS. Provided a child network has a valid architecture[2], it is passed to the iNAS-Exp to find the intermittent execution design that minimizes the *end-to-end inference latency*, else it is assigned a negative reward. The end-to-end inference latency is the total latency to compute the network across multiple power cycles.

The iNAS-Exp performs a joint preservation and execution design space exploration on the child network, while ensuring the feasibility of a candidate design. For each candidate design (i.e., a specific combination of execution and preservation parameters per layer) in the search space, iNAS-Exp uses iNAS-PMod to estimate the energy consumption per power cycle and the end-to-end inference latency. These estimations are used to determine if a candidate design can feasibly execute within the energy budget, without violating the latency requirement. If the lowest achievable end-to-end inference latency by any of the feasible candidate designs is lower than the latency requirement (i.e., $Lat < Lat_{req}$), the DNN trainer is used to train and obtain the accuracy ($Acc$) of the child network. Subsequently, the accuracy and the lowest achievable end-to-end inference latency are added and used as a *reward* signal to update the NAS controller. We formulate the reward similar to an existing HW-NAS [34], where the reward is equal to $(Acc - b) + \frac{Lat}{Lat_{req}}$ when $Lat < Lat_{req}$, else it is equal to $\frac{Lat_{req} - Lat}{Lat_{req}} - 1$, where $b$ is the exponential moving average of the accuracies of the previous feasible child networks. This process is repeated for a specified number of iterations, guiding the search towards a network model that is accurate, safe and efficient. The total search time increases linearly with the number of generated feasible child networks. The resulting *solution* of iNAS is the generated child network with the highest accuracy and its corresponding intermittent execution design which has the lowest end-to-end inference latency.

[2] A valid architecture, for example, is supported by the runtime inference library and has a model size (i.e., weight parameters and OFM buffer requirements) that fits within NVM [58].

### 4.3 Intermittent-aware Execution Design Explorer

*4.3.1 Search Spaces:* The *execution design space* and the *preservation design space* are jointly searched to find, the candidate design that minimizes the end-to-end inference latency. The *execution design space* has five parameters per layer: the tile size ($T_r, T_c, T_m, T_n$) and loop order (denoted $U$), as introduced in Section 2.1. To simplify the latency analysis and the runtime inference implementation, we assume all tiles in a layer are of the same size, and an equal number of tiles are processed in each power cycle. Therefore, the tile size parameters are restricted to be an integer divisor of the layer dimension (i.e., $T_r|R$, $T_c|C$, $T_m|M$ and $T_n|N$, where $R, C, M, N$ denote the layer dimensions). The inter-tile loop order ($U$) can either be the IFM, weight or OFM reuse order (abbrv. *r.o*). POOL layers have an equal number of input and output channels, so $T_m = T_n$ and $U$ = OFM r.o.

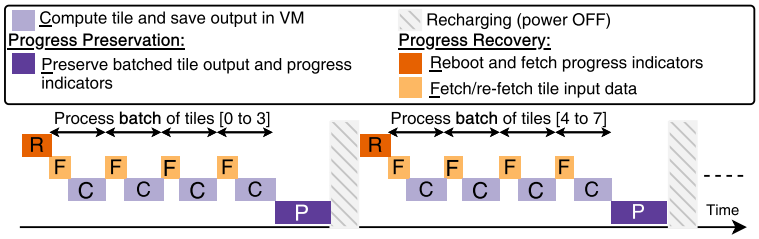


Fig. 5. Intermittent inference execution using batch preservation (e.g., $S$=4)

The *preservation design space* consists of the preservation batch size ($S$) parameter per layer, where $S$ represents the number of tile outputs preserved together in a power cycle (Section 2.2). As shown in Figure 5, for a given layer, the system processes a batch of $S$ tiles in each power cycle, retaining each tile's output in VM. After completing the last tile in the batch, the system preserves the output of all $S$ tiles in NVM and proactively triggers a system shutdown. An equal number of tiles of a layer are processed in each power cycle, so the value of $S$ is restricted to an integer divisor of the number of iterations of the innermost inter-tile loop, which differs depending on the loop order (i.e., $S|\lceil \frac{M}{T_m} \rceil$, $S|\lceil \frac{R}{T_r} \rceil \lceil \frac{C}{T_c} \rceil$ or $S|\lceil \frac{N}{T_n} \rceil$, respectively, if $U$ is equal to the IFM, weight or OFM r.o). Although seemingly straightforward, combining the preservation batch size parameter with the aforementioned parameters in the execution design space is not trivial as it requires considering the compound impact on the preservation and recovery costs, VM usage and computation requirements.

*4.3.2 Feasible Solution Space:* A candidate design is feasible if it satisfies the *energy budget*, *hardware specification* and *inference latency* constraints. The energy budget constraint is required to ensure a batch of tiles can be safely processed without power interruption within a power cycle. Therefore, the energy consumed within each power cycle (denoted $E_{pc}$) by each network layer should be satisfied by the available energy budget per power cycle (denoted $E_{av}$) as follows:

$$\max_{1 \le i \le L} E^i_{pc}(T_r, T_c, T_m, T_n, U, S) \le E_{av} \tag{1}$$

where $E^i_{pc}$ denotes the power-cycle energy consumption of the $i^{th}$ layer in the child network, and $L$ denotes the number of layers in the network. As each layer is assigned specific execution and preservation parameters (Section 4.3.1), the power-cycle energy consumption will vary across the layers. Thus, to satisfy the energy budget constraint, the maximum $E^i_{pc}$ must be less than or equal to $E_{av}$. Here, $E^i_{pc}$ is derived by the iNAS-PMod (Section 4.4) taking the tile size, loop order and preservation batch size as input, considering the key costs related to intermittent inference execution. As discussed in Section 2.2, for a given intermittent system with a capacitor of size

$C_{cap}$ and power ON/OFF voltage thresholds $V_{on}$ and $V_{off}$, the available energy budget $E_{av}$ can be derived as $\frac{1}{2}C_{cap}(V_{on}^2 - V_{off}^2)$.

Hardware specification constraints are required to ensure the candidate design can be feasibly implemented on the target platform with limited resources. Crucially, the VM capacity should be sufficiently large to contain the tile input and output data as follows:

$$B_{in} + B_w + B_{out} \leq VM_{capacity} \tag{2}$$

where $B_{in}$, $B_w$ and $B_{out}$ respectively denote the VM buffer space required for the IFM, weight and OFM tile data. The buffer sizes are related to the tile size, loop order and preservation batch size, where $B_{in} = T_h T_w T_n$ and $B_w = K^2 T_m T_n$. Additional VM space is required to store the output of $S$ tiles, so $B_{out} = S(T_r T_c T_m)$ in all cases, except when the OFM reuse order is used, where the partial sums in the OFM tile buffer are accumulated and overwritten, resulting in $B_{out} = T_r T_c T_m$. Hardware constraints may also include the number of parallel multiply-accumulate (MAC) units available on an accelerator, restricting the tile size parameters $T_m$ and $T_n$ [34, 68].

The inference latency constraint guarantees the user's expected inference timing behavior, such that the given child network when executed intermittently using a specific candidate design will result in an inference latency ($Lat$) lower than the target latency requirement ($Lat_{req}$) as follows:

$$Lat < Lat_{req} \tag{3}$$

where $Lat$ is calculated by the iNAS-PMod (as per Section 4.4), taking the execution and preservation parameters and the dimensions of each network layer, as well as the energy budget as input.

The iNAS-Exp exhaustively evaluates all combinations of the execution and preservation parameters (i.e., the tile size parameters $T_r, T_c, T_m, T_n$, the loop order $U$ and preservation batch size $S$) for each layer in the child network. Therefore, the size of the intermittent-aware execution design space for a layer is $(|T_r| \times |T_c| \times |T_m| \times |T_n| \times |U| \times |S|)$, where the sizes of the discrete parameter sets depend on the layer dimensions (as indicated in Section 4.3.1). The exploration time polynomially increases with the design space. At the end of the design space exploration, the feasible candidate design with the lowest $Lat$ is returned and the child network is trained. The child network is not trained if none of the candidate designs are feasible.

### 4.4 Intermittent-aware Abstract Performance Model

*4.4.1 Progress Preservation Cost:* Intermittent inference execution requires progress preservation to accumulate inference progress across power cycles (Section 2.2). Progress preservation involves preserving *progress indicators* along with the computed *tile outputs* in NVM at the end of each power cycle, and both their associated costs must be included in the total progress preservation cost. Progress indicators can be represented by the four inter-tile loop indices (Section 2.2), and their preservation incurs a fixed cost related to a single NVM write operation. In each power cycle, the tile output of a batch of computed tiles are retained in the OFM tile buffer in VM (with size $B_{out}$, as introduced in Section 4.3.2), and as such, tile output preservation involves writing to specific non-contiguous locations in NVM. Tile output preservation incurs multiple NVM write operations on lightweight systems that commonly use sequential NVM access. The data block size of each NVM write operation and the number of such operations required to preserve the tile output will vary based on the tile size, preservation batch size and inter-tile loop order. Therefore, the total progress preservation cost (denoted $pp$) can be estimated as follows:

$$pp = \begin{cases} T_r T_c[write(ST_m)] + write(4), & \text{if } U = \text{IFM r.o} \\ ST_r T_c[write(T_m)] + write(4), & \text{if } U = \text{Weight r.o} \\ T_r T_c[write(T_m)] + write(4), & \text{if } U = \text{OFM r.o} \end{cases} \tag{4}$$

where $write(z)$ refers to a single NVM write operation with a data block size $z$. The derivation of the preservation and recovery costs (Eq. 4 and Eq. 5) may be simpler for systems with sophisticated memory access capability (e.g., 2D DMA support), requiring fewer write operations for tile input/output data transfers. However, lightweight MCUs (e.g., MSP430 or Arm Cortex-M0/M4 MCUs) such as those commonly used in intermittent systems, typically do not support such features.

*4.4.2 Progress Recovery Cost:* Intermittent inference execution requires progress recovery at the start of every power cycle to correctly resume the interrupted inference process. Progress recovery involves *system reboot*, fetching the *progress indicators* and *tile input data* from NVM to VM, and all associated costs must be included in the total progress recovery cost. System reboot incurs a fixed cost of peripheral initialization (e.g., the accelerator and NVM communication interface) and fetching the index of the currently computing network layer from NVM. Fetching the progress indicators also incurs a fixed cost of reading the four inter-tile loop indices from NVM using a single read operation. Tile input data need to be re-fetched in each power cycle, as the data fetched into VM in the previous power cycle are lost due to power loss, which is a characteristic of intermittent inference. Figure 6 illustrates an example where under continuous power inference, using the IFM reuse loop order, the IFM tile data are fetched in the first tile and reused across subsequent tiles. However, under intermittent power, if batch preservation is not considered (i.e., $S=1$), the IFM tile data cannot be reused, as the data are lost at the end of a power cycle, and therefore the data need to be re-fetched in the next power cycle. Thus, maximizing data reuse, as performed by HW-NAS, may be inefficient under intermittent power, as it may cause a higher data re-fetching cost. In contrast, when batch preservation is considered (i.e., $S>1$), the respective IFM, weight or OFM tile input data fetched in the first tile can be reused across all the subsequent tiles computed in the current power cycle (Figure 6 bottom). Correctly considering the progress recovery cost (as formulated in Eq. 5) effectively allows iNAS-Exp to find a candidate design which can balance the data reuse benefit and data re-fetching cost.

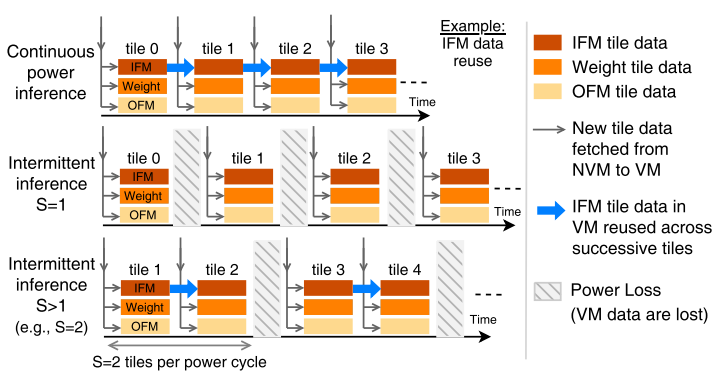


Fig. 6. Data reuse under continuous and intermittent inference (when preservation batch size $S=1$ and $S>1$)

The cost of fetching the tile input data within a power cycle will differ based on the loop order, the tile size and preservation batch size, and therefore the overall progress recovery cost (denoted $pr$) can be estimated as follows:

$$pr = \begin{cases} rbt + read(4) + \left[S(\Gamma_w + \Gamma_o) + \Gamma_i\right], & \text{if } U = \text{IFM r.o} \\ rbt + read(4) + \left[S(\Gamma_i + \Gamma_o) + \Gamma_w\right], & \text{if } U = \text{Weight r.o} \\ rbt + read(4) + \left[S(\Gamma_i + \Gamma_w) + \Gamma_o\right], & \text{if } U = \text{OFM r.o} \end{cases} \tag{5}$$

where $rbt$ is the reboot cost and $read(z)$ refers to a single NVM read operation with data block size $z$. Also, $\Gamma_i$, $\Gamma_w$ and $\Gamma_o$ respectively denote the IFM, weight and OFM tile input data fetching costs which are calculated as $\Gamma_i$=$T_{ri}T_{ci}[read(T_n)]$, $\Gamma_w$=$K^2T_m[read(T_n)]$ and $\Gamma_o$=$T_rT_c[read(T_m)]$, assuming sequential NVM access.

*4.4.3 Computation Cost:* As intermittent inference employs batch preservation, the total cost of computing multiple tiles in a power cycle needs to be considered. Computing one tile may require successive accelerator operation executions on resource constrained systems that often only support simple vector math hardware acceleration [5, 26]. On such lightweight accelerators, the innermost intra-tile loop (Section 2.2) can be unrolled and executed as a vector multiply-accumulate (MAC) operation followed by a scalar addition. Here, we adapt the computation cost formulation used for architectural accelerators [34, 56, 68], to support lightweight accelerators and batch tile processing. Hence, the total computation cost (denoted $cp$) in a power cycle to compute a batch of $S$ tiles can be estimated as follows:

$$cp = S\Big(K^2T_rT_cT_m[vecMAC(T_n) + Add]\Big) \tag{6}$$

where $vecMAC$ refers to the vector MAC operation between the weight kernel and the IFM, of length $T_n$ in the input channel dimension, and $ADD$ refers to the scalar addition of the current and previously computed partial sums. Both computations are executed $K^2T_rT_cT_m$ times per tile, and $S$ tiles are computed in a power cycle. Larger accelerators may unroll and execute multiple intra-tile loops in parallel [56, 68], reducing the number of operation executions required per tile (in Eq. 6).

*4.4.4 Power-cycle Energy Consumption and Latency:* The power-cycle energy consumption ($E_{pc}$) and latency ($L_{pc}$) of a specific candidate design are respectively required to determine if the design satisfies the energy budget constraint (Section 4.3.2) and to derive its end-to-end inference latency (Section 4.4.6). To calculate $E_{pc}$ and $L_{pc}$, the progress preservation, progress recovery and computation costs of a power cycle (defined in Eq. 4, 5 and 6) are added together as follows:

$$E_{pc} = E(pp) + E(pr) + E(cp) \tag{7}$$

$$L_{pc} = L(pp) + L(pr) + L(cp) \tag{8}$$

In Eq. 7 and Eq. 8, $E()$ and $L()$ respectively refer to the energy consumption and latency of the aforementioned $pp$, $pr$ and $cp$ costs (defined in Eq. 4, 5 and 6). For example, by E(pp) and L(pp) we respectively refer to the energy E() and latency L() costs of progress preservation (pp). In a candidate design, each layer has specific execution and preservation parameters, so $E^i_{pc}$ and $L^i_{pc}$ are used to denote the power-cycle energy consumption and latency of the $i^{th}$ layer. An equal number of tiles in a layer are processed in each power cycle (Section 4.3.1), so the energy and latency costs of each power cycle are equal.

*4.4.5 Recharge Time:* The recharge time (denoted as $L_{rc}$) is the period in which the harvested energy is accumulated into the energy buffer, and $L_{rc}$ is required to derive the end-to-end inference latency. As illustrated in Figure 7, the energy buffer is discharged during execution and recharged after the system is proactively shut down at the end of a power cycle. Depending on the amount of energy consumed in the power cycle (i.e., $E_{pc}$ as formulated in Eq. 7) before the system is shut down, recharge can start from a partially full energy buffer. Therefore, $L_{rc}$ is directly proportional to the energy buffer size ($C_{cap}$) and $E_{pc}$, and can be expressed as follows:

$$L_{rc} = -R_{eq}C_{cap}\ln\left[\frac{V_{on} - V_{sup}}{V_{pc} - V_{sup}}\right] \tag{9}$$

where $V_{sup}$ is the supply voltage, $R_{eq}$ is the equivalent resistance of the energy harvesting and power management circuitry and $V_{on}$ is the system power ON voltage threshold (Section 2.2). As

shown in Figure 7, the voltage across the energy buffer at the end of a power cycle after $E_{pc}$ has been consumed is denoted as $V_{pc}$, and is equal to $\sqrt{2(E_{av} - E_{pc})/C_{cap}} + V_{off}$.

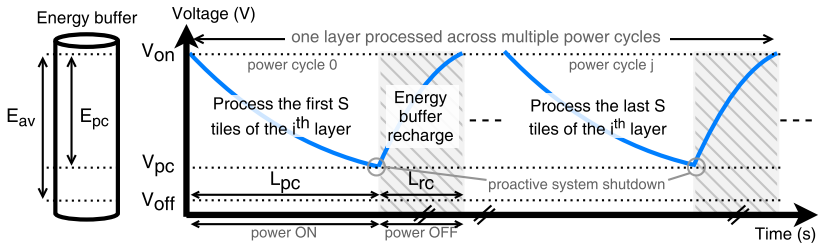


Fig. 7. Recharge and discharge phases of the energy buffer across power cycles

*4.4.6 End-to-end Inference Latency:* The total time taken to complete an end-to-end inference of a child network across multiple power cycles is the summation of the power-cycle latency (i.e., $L_{pc}$ as given in Eq. 8) and the energy buffer recharge duration (i.e., $L_{rc}$ as given in Eq. 9) of all power cycles, across all layers in the network. Recall that $L_{pc}$ includes all latency costs within a power cycle and $L_{rc}$ relies on the energy consumption of a power cycle. Thus, the end-to-end inference latency ($Lat$) can be calculated as follows:

$$Lat = \sum_{i=1}^{L} \sum_{j=1}^{N_{pc}} \left( L_{pc}^{i,j} + L_{rc}^{i,j} \right) \tag{10}$$

where $L_{pc}^{i,j}$ and $L_{rc}^{i,j}$ respectively denote the power-cycle latency and the recharge duration, corresponding to the $j^{th}$ power cycle of the $i^{th}$ layer in the child network (Figure 7). In Eq. 10, $N_{pc}$ denotes the number of power cycles required to complete a layer and is calculated as $\left(\lceil \frac{R}{T_r} \rceil \lceil \frac{C}{T_c} \rceil \lceil \frac{N}{T_n} \rceil \lceil \frac{M}{T_m} \rceil / S\right)$. Therefore, larger tile sizes and preservation batch sizes result in fewer power cycles required to complete an inference, provided the energy budget constraint (Eq. 1) is not violated.

## 4.5 Generality

Although the proposed intermittent-aware execution design explorer and abstract performance model (in Sections 4.3 and 4.4) primarily consider CONV layers, other common types of DNN layers are also supported with trivial or no adaptations. FC layers can be formulated as CONV layers by setting the weight kernel dimensions to match the IFM dimensions, and POOL layers can also be tiled, where multiple pooling windows can be processed within a tile. Furthermore, advanced models with skip connections and feedback paths [58] can also be supported by including the additional data fetching costs in the progress recovery cost. iNAS can also be extended to support other intermittent inference execution models such as task-based models without proactive shutdown [19] and footprint-based models [36]. To support the former, the progress recovery cost formulation (Section 4.4.2) should be extended to capture the cost of inference re-execution. To support the latter, the progress preservation cost formulation (Section 4.4.1) should be extended to include the costs of fine-grain progress and inference output preservation.

iNAS is also compatible with other NAS frameworks that contain additional design spaces. For example, the DNN compression space (e.g., filter/channel pruning and weight quantization) [32] can be integrated into iNAS, with modifications to the performance model to consider the impact on the energy and latency cost due to different compression parameters. Furthermore, we demonstrate our proposed preservation space can already offer significant intermittent inference improvements over HW-NAS (Section 6.2.1), but our methodology can be leveraged by others

to expand the preservation space or combine it with a compression space to achieve further improvements.

As iNAS assumes a generic tile-based intermittent inference execution model, we expect iNAS can directly support widely available, ultra-low power MCU architectures (e.g., TI MSP430 and ARM Cortex-M0/M4), which are commonly used by intermittent systems [60]. However, it is important to note that for intermittent systems, the selection of the appropriate VM and capacitor sizes is more crucial than the type of MCU architecture, as the cost of data transfer between the VM and NVM, and the energy buffer recharge cost significantly affects intermittent execution performance. iNAS can be also extended to support other types of intermittently-powered systems (e.g., FPGA-based [69] or multiprocessor intermittent systems [57]), by considering any additional preservation and recovery costs related to those specific systems. These extensions may not be trivial, and thus require further investigation, where iNAS can provide a foundation for such future research.

## 5 INAS IMPLEMENTATION

iNAS was implemented on top of an existing HW-NAS framework [34] using Tensorflow 1.15 and executed on an NVIDIA GTX 1080 Ti GPU. The solutions found by iNAS were deployed on the Texas Instruments (TI) MSP430FR5994 [27] MCU, with 16 MHz clock speed, a TI Low Energy Accelerator (LEA) and 8KB SRAM (2KB dedicated for accelerated inference). External NVM (Cypress CY15B108Q 1MB serial FRAM [14]) was used to support reasonably large DNNs.

### 5.1 Accelerated Intermittent Inference

A hardware-accelerated intermittent inference library was developed for the TI platform to support tiled DNN processing and task-based intermittent execution with proactive shutdown. The LEA vector multiply-accumulate command (i.e., `LEACMD__MAC` with vector size $T_n$) was used to compute the innermost intra-tile loop (Section 4.4.3), and for compatibility with the LEA, the tile size parameter $T_n$ was restricted to one or an even value. External NVM was interfaced via SPI (serial peripheral interface) and was configured to co-operate with the DMA (direct memory access) controller to transfer tile input/output data one byte at a time, without CPU intervention. The iNAS solutions are represented using custom data structures defined by the inference library. For compatibility with the MSP430 platform, all models were compressed by quantizing the input and weight parameters to a 16-bit fixed point representation (Q15.1 format) from the 32-bit floating point representation used during training, without significant loss of accuracy.

To ensure *idempotency*, where repeated execution does not produce a new result each time, the inference library avoids write-after-read dependencies by using a double buffering mechanism similar to that in [19, 50] when reading and writing to the OFM buffer in NVM. After processing a batch of tiles, the system is proactively shut down (as discussed in Sections 2.2 and 4.3.1) by employing an external power control switch similar to [31, 35]. The intermittent inference library sends a signal to the power control switch to close the power supply path to the MCU and start the energy buffer recharge once a batch of tiles is completed. An additional control switch at the entry to the energy management unit ensures the capacitor recharge and system execution processes are separated, and also ensures the system is solely driven by the capacitor energy.

### 5.2 Energy and Latency Estimation

A set of micro-benchmarks were developed to profile the low-level energy and latency costs of the TI platform, allowing us to derive the costs related to intermittent inference (i.e., progress preservation, progress recovery and computation) discussed in Section 4.4. We used linear regression to model the cost of the `LEACMD__MAC` command with respect to the input vector size (i.e., *vecMAC* operation

in Eq. 6) and the cost of a DMA-based NVM read/write transfer with respect to the data block size (i.e., *read* and *write* operations in Eq. 4 and 5). These operations incur an additional fixed overhead related to the operation input/output memory address pointer calculation (e.g., calculating the source/destination memory location for an NVM *read*/*write* operation). Therefore, we also obtained latency and energy measurements of fine-grain CPU-based math instructions (e.g., add, multiply, divide etc.), to allow us to estimate the pointer address calculation overhead of each operation invocation. In addition, each DMA transfer incurs a DMA invocation of 2 clock cycles and an NVM write overhead of 16 clock cycles. All energy and latency measurements were respectively obtained using the TI EnergyTrace software [25] and a hardware timer in the MCU. Due to physical characteristics such as capacitor leakage, charge redistribution and voltage ripples in the power management circuitry [3], the actual runtime energy budget may vary slightly from the estimate ($E_{av}$). Therefore, to guarantee safe inference execution, the power-cycle energy budget is underestimated by 45%, which was derived using a measurement-based approach on our prototype platform. This safety margin can be further reduced to allow iNAS to find intermittent execution designs that can better utilize the energy budget, but at increased risk.

## 6 EVALUATION

### 6.1 Experimental Setup

We evaluated the amount of improvement iNAS brings to the HW-NAS framework [34][3]. The solutions of both iNAS and HW-NAS with the highest accuracy across multiple runs are deployed and evaluated on the TI MSP430FR5994 platform, as described in Section 5. Intermittent execution was emulated using a Keithley 2280S power supply, a TI-BQ25570 energy harvesting and management unit and a capacitor. A power supply of 6 mW (2 V, 3 mA) was used, representative of indoor light energy [48], which is insufficient to continuously operate the TI platform. Also, the system power ON/OFF thresholds were respectively set at 3V and 2.8V.

Table 1. Evaluation datasets and related parameter settings

| Dataset | DNN Trainer | | NAS Controller | | |
|---|---|---|---|---|---|
| | Training | Validation | Model architecture | K | M |
| CIFAR-10 | 50000 | 10000 | 5 × 2D CONV, GPOOL, FC | 1,3,5,7 | 4,8,16,24 |
| HAR | 7400 | 3000 | 3 × 1D CONV, GPOOL, FC | 1,3,5,7 | 4,8,16,24 |
| KWS | 37000 | 4900 | 5 × FC | - | 48,64,96,128 |

CONV: convolutional layers, GPOOL: global average pooling, FC: fully connected layer, K: kernel size, M: number of filters

Table 1 shows the three datasets used for evaluation, taken from the tinyML benchmark suite [6] and representative of tiny machine learning applications. These include an image classification dataset [38] referred to as CIFAR-10, an accelerometer sensor dataset [4] used for human activity recognition referred to as HAR and a speech keywords dataset [62] referred to as KWS. Each dataset is composed of a training and validation set. A feasible child network is trained for 25 epochs, with the maximum validation accuracy used to compute the reward. Different NAS controller settings and model architecture types are used to suit different datasets. For example, the architecture for CIFAR-10 (in Table 1) has five 2D CONV layers (i.e., dense convolution with stride=1 and no zero-padding), a global POOL and an FC layer, and for each network layer, the NAS controller selects the kernel size and the number of filters from a given selection (i.e., $K$=1,3,5,7 and $M$=4,8,16,24).

[3]The HW-NAS framework [34] was originally developed for FPGA-based systems. For fair comparison, we have adapted it to support MCU-based systems.

Therefore, the size of the model architecture space is 1048576, 4096 and 256, respectively for the CIFAR-10, HAR and KWS datasets. The size of the intermittent-aware execution and preservation design space for a generated child network depends on its layer dimensions, where for the above settings the maximum design space size is 846336, 71904 and 13464 respectively for the CIFAR-10, HAR and KWS datasets. For each dataset, 100 child networks are generated (i.e., 100 iterations).

To evaluate the ability of iNAS and HW-NAS to find feasible solutions for different intermittent execution scenarios, we conduct experiments under small, medium and large energy budgets, respectively specified by the capacitor sizes 1 mF, 5 mF and 10 mF. Given the above platform configuration, these correspond to 0.55 mJ, 2.7 mJ and 5.5 mJ energy budgets. Unlike iNAS, HW-NAS does not consider the intermittent system energy budget, so the same HW-NAS solution is evaluated under different capacitor sizes. To further observe the efficacy of the derived solutions under different feasible solution spaces, we evaluate iNAS and HW-NAS under three different inference latency requirements ($Lat_{req}$), denoted as TS1 (low), TS2 (medium) and TS3 (high), and they are respectively set such that 25%, 50% and 75% of the solutions are feasible in the search space. While ensuring these same percentage ratios, we use separate latency requirements per capacitor size and dataset as well as for HW-NAS. This is because the energy budget and model architecture type directly impact the end-to-end inference latency of the iNAS solutions, and HW-NAS requires different latency requirements for fair comparison as it does not consider the additional costs related to intermittent inference execution across multiple power cycles.

In our primary set of experiments, we evaluate the resulting solutions of iNAS and HW-NAS in terms of their *end-to-end inference latency* on the TI platform under intermittent power, their model *accuracy* on the dataset and the average *search time* across multiple NAS runs. As discussed in Sections 4.3 and 4.4, the *VM capacity size* and the *energy budget* of an intermittent system are key factors that affect the costs related to intermittent execution behavior, and thus impact the end-to-end inference latency. Therefore, we conduct a secondary set of experiments to provide useful design suggestions for selecting an appropriate VM capacity size and capacitor size for a new intermittent inference design. For these secondary experiments, we use the network model solutions obtained by iNAS from the primary experiments (i.e., under the 10 mF and TS3 cases, for each dataset) and run iNAS-Exp and iNAS-PMod under varied capacitor and VM sizes.

### 6.2 Performance Comparison

*6.2.1 Inference latency:* Figure 8 shows the end-to-end intermittent inference latency on the TI platform of the solutions found by iNAS and HW-NAS, across all datasets, capacitor sizes and latency requirements. The results indicate that the HW-NAS solutions may be *unsafe*, as they cannot complete the inference when the energy budget is insufficient. For example, the HW-NAS solutions fail to make forward progress under the small capacitor (1 mF) in all three datasets, as well as in several cases under the medium capacitor (e.g., in CIFAR-10 when the latency requirement was set to TS2 and TS3). This is primarily because HW-NAS maximizes data reuse by finding execution designs consisting of large tile sizes with higher energy cost, without considering the power-cycle energy budget. In contrast, iNAS ensures the power-cycle energy consumption of a solution can be satisfied by the intermittent system, and therefore the iNAS solutions can safely complete the inference under all evaluated scenarios, even under the small energy budget.

Although the HW-NAS solutions can complete the inference when the energy budget is sufficient, the solutions are *inefficient* in all such cases, as they do not meet the end-to-end latency requirement. This is particularly evident when the latency requirement is higher (e.g., in CIFAR-10 under 10 mF/TS3 and in HAR under 5 and 10 mF/TS3), as HW-NAS finds more complex solutions, such as network architectures with a larger number of filters and larger kernel sizes. Because execution designs are constrained by the VM capacity size and HW-NAS does not consider batch

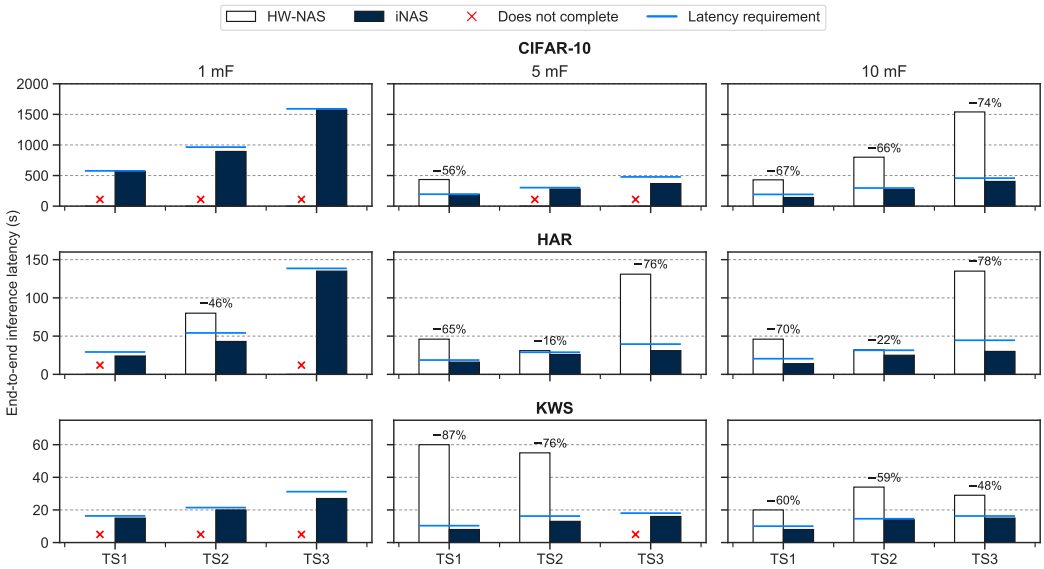


Fig. 8. End-to-end inference latency of iNAS solutions compared to HW-NAS

preservation and additional costs related to intermittent execution behavior, the derived complex solutions require a large number of power cycles to complete, while significantly underutilizing the power-cycle energy budget. Additionally, HW-NAS solutions are susceptible to re-execution, further increasing their end-to-end inference latency. Re-execution is more noticeable under a marginally sufficient energy budget that shows slight variation at runtime due to the capacitor's physical characteristics (Section 5.2), and also when the power-cycle energy cost per layer has lower variation due to a small architecture search space (e.g., KWS under 5 mF). In contrast, iNAS considers batch preservation and appropriately balances data reuse and costs related to intermittent execution, and therefore the derived solutions allow for better utilization of the energy budget and require fewer power cycles to complete inference. Therefore, the iNAS solutions can efficiently meet the end-to-end latency requirement under all experiment conditions. Moreover, as iNAS underestimates the energy budget (Section 5.2), slight variation leading to re-execution are overcome.

The relative reductions in the inference latency of the iNAS solutions compared with the HW-NAS solutions are indicated above the bar plots. Overall, compared to the HW-NAS solutions that can safely complete the inference, the iNAS solutions reduce the intermittent inference latency by 16% to 87% (on average 60%). The improvements are more apparent when it is especially challenging for HW-NAS to find feasible solutions, for instance under a small capacitor, where the inference does not complete, and additionally under a high latency requirement, where more complex network architectures are found.

*6.2.2 Solution Accuracy:* Figure 9 shows the relative accuracy difference of the solutions found by iNAS with respect to the HW-NAS solutions, for each dataset under different capacitor sizes and latency requirements. iNAS shows an accuracy gain of up to 3.4% (e.g., CIFAR-10 under 1 mF) but also an accuracy loss of up to 1.2%, 0.82% and 0.4%, respectively, for the CIFAR-10, HAR and KWS datasets. However, note that iNAS by design does not improve or degrade accuracy, and the highest accuracy found by any of the explored solutions is highly dependent on the size of

the feasible solution space, controlled by the latency requirement [34]. The accuracy difference between the HW-NAS and iNAS solutions are mainly contributed by the DNN trainer, as the accuracy of the same child network may vary 1-2% each time it is trained. Compared to the HAR and KWS datasets, the accuracy difference is higher in the CIFAR-10 dataset, because its network model type has a larger architecture search space, and hence the number of filters and kernel size vary greatly across the explored child networks.

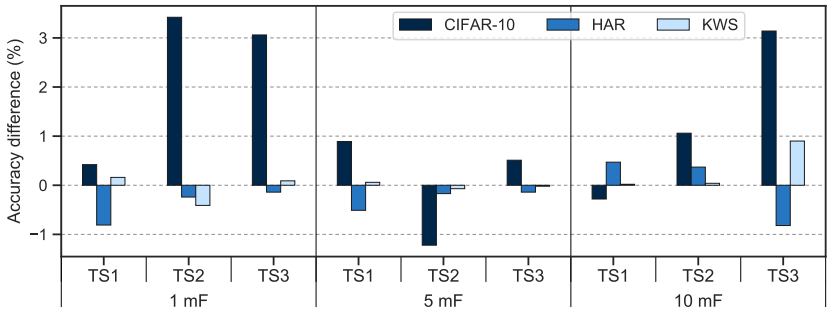


Fig. 9. Relative accuracy difference of iNAS solutions compared to HW-NAS solutions

*6.2.3 Search Time:* Figure 10 shows the difference between the search time of iNAS compared with HW-NAS, averaged across multiple runs. The bars in the figure indicate the relative search time difference, and the absolute search time of each framework is reported above the bars (displayed in minutes, rounded to the nearest integer; top: iNAS, bottom: HW-NAS). The search time is dominated by the training time of the feasible child networks, and therefore the CIFAR-10 and HAR datasets respectively have the highest and lowest search time, considering the size of their training data and their model architecture type. Moreover, a higher latency requirement also increases the search time as it increases the number of feasible child networks which are trained.

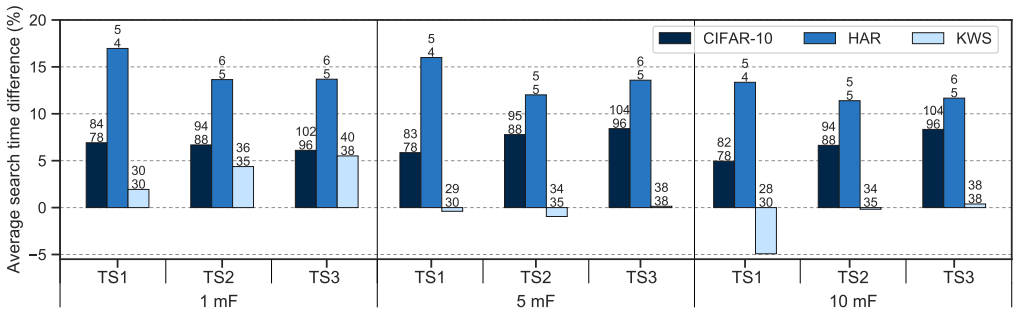


Fig. 10. Relative search time difference of iNAS compared to HW-NAS

Compared to HW-NAS, iNAS incurs extra overhead to explore the preservation design space in addition to the execution design and network architecture spaces, and to calculate the power-cycle energy consumption and latency which are used to estimate the end-to-end inference latency. This additional overhead is more apparent for datasets with less training data, such as HAR, where the iNAS search time is 11% to 17% higher than HW-NAS. By contrast, for datasets with a relatively large training set such as CIFAR-10 and KWS, the iNAS overhead is respectively only 5% to 8%

and 0.1% to 5.5% higher than HW-NAS. In a few cases iNAS finds marginally fewer feasible child networks than HW-NAS, resulting in a decrease in the overhead (e.g., KWS under 10 mF and TS1).

### 6.3 Constraint Characterization

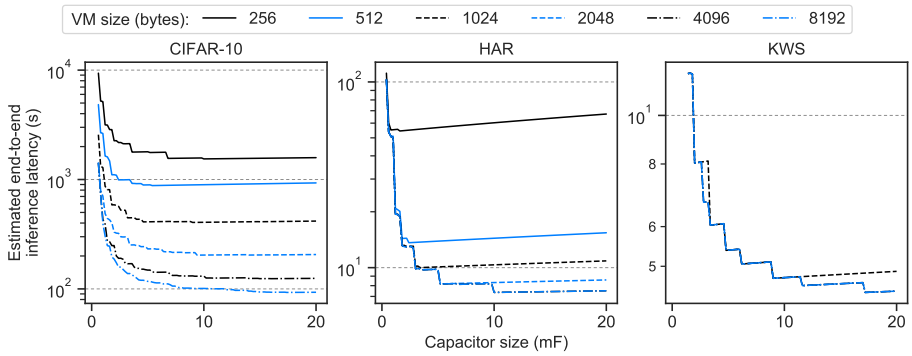


Fig. 11. Impact of VM size and capacitor size on end-to-end inference latency

Figure 11 shows the impact of the capacitor size and VM size on the end-to-end inference latency, explored across the three model architecture types used for the evaluation datasets. We make several key observations from this analysis. Firstly, the VM and capacitor sizes are tightly coupled. Larger VM sizes allow for execution designs with larger tile sizes, reducing the total data access costs and lowering the inference latency. However, after a certain point, doubling the VM size does not significantly improve the inference latency, as the energy budget becomes a bottleneck, restricting the preservation batch size. Secondly, for model architectures with a smaller search space, such as KWS, larger VM sizes do not bring any noticeable latency improvement, for the same capacitor size. This is because, for such model architecture types, under different VM sizes, the intermittent execution design that gives the lowest latency may require an equal number of power cycles to complete, resulting in a low variation in end-to-end inference latency. Thirdly, increasing the capacitor size does not necessarily reduce the end-to-end inference latency. A larger capacitor size allows for a larger preservation batch size, which reduces the number of required power cycles for intermittent inference and improves the end-to-end inference latency. However, for a given VM size, after the maximum feasible preservation batch size has been reached, the capacitor recharge time will be a bottleneck, which increases with the capacitor size, and thus impacts the end-to-end inference latency. This is clearly seen in HAR under small VM sizes (e.g., 256 and 512 bytes), because the HAR model architecture type has a smaller search space than that of CIFAR-10 and a larger input size than that of KWS. This quickly saturates the VM capacity, given a large intermittent execution design.

## 7 DISCUSSION

Although through extensive experiments we demonstrate the practicality of iNAS, there are several opportunities for further performance improvement. The iNAS framework does not target a specific type of application, instead iNAS is generic and broadly applicable for intermittent inference systems typically running lightweight DNNs. As shown in Section 6.3, iNAS is also well suited as an early design exploration and system evaluation tool, to facilitate key platform design choices during early design stages. System design considerations should be appropriately determined based on the application requirements. For example, if low latency is a critical requirement, then

the system can be designed with a larger capacitor size or stronger power source (at a higher area and component cost), to respectively reduce the number of power cycles or recharge time. Alternatively, techniques such as model compression and runtime adaptation can also be integrated to reduce the inference latency (Section 8)

In this first work, exhaustively exploring the execution and preservation design spaces is simple yet manageable. However, exhaustive search will become infeasible as the network size and design space becomes larger, and therefore more efficient and sophisticated exploration algorithms may need to be used (e.g., evolutionary search [51], heuristic-based speed-ups [40] or dynamic programming approaches [46]). Similarly, as discussed in Section 8, faster architecture search strategies such as gradient-based optimization can also be applied to speed up the NAS.

As discussed in Section 5.2, the proposed intermittent-aware abstract performance model (iNAS-PMod) is general and broadly applicable to intermittently-powered edge devices characterized in Section 2.1, although the target devices need to be profiled. In order to calibrate iNAS-PMod, we obtain energy/latency measurements of micro computations and data transfer operations on the target intermittent platform, as explained in Section 5.2. Minor calibration may be required to determine the equivalent resistance of the energy harvesting unit and power management circuits, if not stated in the specification. Nevertheless, please note that abstract performance models typically rely on some form of calibration; for example, using information in datasheets, simulator traces [53], real hardware measurements to train machine learning models [9], or similar to our approach, calibration may be in the form of profiling low-level operations [19]. Moreover, the process of calibration can be automated [18], and other modeling approaches that do not rely on per-device calibration can also be adopted to improve iNAS-PMod (e.g., [2, 11, 64]). In the performance model, it is not crucial for the estimated absolute performance of a solution to be highly accurate; instead iNAS requires the *relative order* of the solutions in the search space to be accurate, in terms of their relative performance, to find the solution with the lowest end-to-end inference latency. In additional, to ensure safe execution while experiencing energy budget variation at runtime, iNAS underestimates the power-cycle energy budget, where the safety margin can be increased to further improve energy budget utilization, but at the risk of failure.

## 8 RELATED WORK

HW-NAS frameworks incorporate hardware-aware objectives and constraints, such as inference latency [34, 65], energy consumption [15, 51] and resource usage [66] into the NAS optimization, to find networks with high accuracy and hardware performance. To further maximize hardware efficiency, the DNN architecture and hardware design spaces are merged and simultaneously co-searched [1, 66]. HW-NAS fundamentally maximizes data reuse to achieve low inference latency, while minimizing resource and energy usage. HW-NAS may enforce an energy constraint on the total inference energy consumption [15, 51], but not at the power-cycle level, so safe execution cannot be guaranteed. Furthermore, none of the existing techniques explore any form of preservation design space, which may lead to sub-optimal designs for intermittent systems. In contrast, our work is the first to introduce intermittent execution behavior into NAS, by enabling NAS to appropriately balance data reuse and costs related to intermittent execution, and to co-search a new preservation design space unique to intermittent inference, while ensuring solutions do not exceed the power-cycle energy budget. Smaller and faster networks with low accuracy loss can be found by exposing model compression (i.e., minimization) spaces of techniques such as hardware-aware pruning [32, 43], where parameter redundancy is reduced, quantization-aware training [20, 61], where small bit-width models are trained, and knowledge distillation [54], where knowledge from a larger model is transferred to a smaller model. On an intermittent system, where the power source is weak and unstable, in most cases, a compressed model would still

require multiple power cycles to complete an inference. Therefore, compression may still have to be combined with design-time tools or runtime software to guarantee safe execution. Future efforts can leverage our methodology to integrate these orthogonal compression spaces into intermittent-aware NAS to achieve better inference latency improvements.

There is a growing interest to speed up NAS, as it can become extremely time consuming, especially when considering wide design spaces or large datasets [59, 70]. Hierarchically combined micro and macro architecture spaces reduce the search space size, while maintaining a high degree of search freedom [45, 59]. Concepts such as weight sharing [8, 9, 44] and accuracy predictors [63], alleviate the cost of training individual child networks. Fast gradient-based search optimizations make the NAS objective function differentiable [12, 41], but special transformations are required for non-differentiable metrics such as inference latency [9]. In contrast, search strategies such as reinforcement learning [70] and evolutionary algorithms [51, 55], although computationally expensive, are generic and support architectures with a broad range of criteria. Interested readers can find a comprehensive review of NAS techniques (including HW-NAS) in [7, 16]. We consider a macro architecture space with key parameters and a generic reinforcement learning search strategy that is commonly used by HW-NAS [33, 34, 59]. These are sufficient to demonstrate the effectiveness of our approach to produce safe and efficient solutions for intermittent systems. Our framework can indeed be complemented by integrating any of the above mentioned NAS speed up techniques.

Initial work on intermittent inference provided task-based runtime software to reduce the overhead of progress preservation and recovery of loop-heavy inference computations [19]. Our framework assumes a similar execution model as in [19], extended with a proactive system shutdown strategy. Hardware accelerated inference operations can also be accumulatively executed across power cycles, by employing fine-grain progress tracking, which exploits the inherent operational behavior of lightweight accelerators found in commercial off-the-shelf MCUs [36]. Runtime system software may also opportunistically adapt the inference to vary the inference performance depending on the harvested energy level. Existing adaptation techniques may grow/shrink network elements and perform runtime retraining with the help of a co-processor [39], or select early termination points on a multi-exit network without significant accuracy degradation, by using a power trace-aware offline optimization [42, 67]. Intermittent inference can also be applied to applications with real-time requirements and continuous data (e.g., audio keyword recognition) by combining early termination techniques and energy-aware scheduling [29, 30]. In essence, the above intermittent inference techniques and runtime adaptation techniques efficiently execute or incrementally adapt a fixed baseline DNN model, without searching for an optimum architecture for the target intermittent platform. Extending our work to consider these efficient runtime methods may open up new challenges in intermittent-aware NAS.

## 9 CONCLUDING REMARKS

This paper presents iNAS, which introduces intermittent execution behavior into NAS, to find accurate network architectures with corresponding intermittent execution designs that can be feasibly deployed on intermittently-powered systems. Existing NAS frameworks find inference execution designs that maximize data reuse, which under intermittent power can be inefficient as they may not meet the end-to-end intermittent latency requirement and, even more seriously, the execution designs may be unsafe as they may not safely make forward progress if the power-cycle energy budget is insufficient. In contrast, iNAS is able to find intermittent execution designs which are both safe and efficient by using an intermittent-aware execution design explorer and an

intermittent-aware abstract performance model. The solutions found by iNAS[4] and an existing HW-NAS framework [34] were evaluated on a Texas Instruments device under intermittent power. In all scenarios, the iNAS solutions safely meet the latency requirements and show improved end-to-end inference latency compared to the HW-NAS solutions. The improvements are more evident when it is harder for HW-NAS to find feasible solutions, such as when the energy budget is small, as well as when the latency requirement is high, where HW-NAS fails to find efficient execution designs for complex network architectures. Furthermore, the iNAS solutions show accuracy comparable to the HW-NAS solutions, and iNAS incurs an acceptable increase in search overhead.

The iNAS framework and the intermittent inference library, has been made openly available [52], allowing the automatic design and deployment of DNNs with high accuracy and low latency for energy harvesting edge devices, and additionally, iNAS can also assist in the selection of appropriate configurations for a new intermittent platform to provide the required performance.

## REFERENCES


[1] Mohamed S Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D Lane. 2020. Best of Both worlds: AutoML Codesign of a CNN and its Hardware Accelerator. In *Proc. of ACM/IEEE DAC*. 1–6.

[2] Saad Ahmed, Muhammad Nawaz, Abu Bakar, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2020. Demystifying Energy Consumption Dynamics in Transiently powered Computers. *ACM TECS* 19, 6 (2020), 1–25.

[3] JunIck Ahn, Daeyong Kim, Rhan Ha, and Hojung Cha. 2021. State-of-Charge Estimation of Supercapacitors in Transiently-powered Sensor Nodes. *IEEE TCAD* (2021).

[4] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L. Reyes-Ortiz. 2013. Dataset for Human Activity Recognition using Smartphones. https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones.

[5] ARM. 2021. Cortex M4 DSP ISA. https://developer.arm.com/architectures/instruction-sets/dsp-extensions/dsp-for-cortex-m.

[6] Colby R Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, et al. 2020. Benchmarking TinyML Systems: Challenges and Direction. *arXiv preprint arXiv:2003.04821* (2020).

[7] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. 2021. A Comprehensive Survey on Hardware-Aware Neural Architecture Search. *arXiv preprint arXiv:2101.09336* (2021).

[8] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *Proc. of ICLR*.

[9] Han Cai, Ligeng Zhu, and Song Han. 2018. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *Proc. of ICLR*.

[10] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE JSSC* 52, 1 (2016), 127–138.

[11] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. 2017. Determining Application-Specific Peak Power and Energy Requirements for Ultra-low Power Processors. In *Proc. of ACM ASPLOS*. 3–16.

[12] Kanghyun Choi, Deokki Hong, Hojae Yoon, Joonsang Yu, Youngsok Kim, and Jinho Lee. 2021. DANCE: Differentiable Accelerator/Network Co-Exploration. In *Proc. of ACM/IEEE DAC (Accepted)*. IEEE, 1–6.

[13] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proc. of ACM ASPLOS*. 767–781.

[14] Cypress. 2019. Excelon LP 8-Mbit SPI F-RAM. https://www.cypress.com/file/444186/download.

[15] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, and et al. 2019. ChamNet: Towards Efficient Network Design Through Platform-Aware Model Adaptation. In *Proc. of IEEE CVPR*. 11398–11407.

[16] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. 2019. Neural Architecture Search: A Survey. *Journal of Mach. Learn. Res.* 20, 55 (2019), 1–21.


[4]Interested readers may refer to a demo at https://youtu.be/XJhoIHSVCOo

[17] Eric Flamand, Davide Rossi, Francesco Conti, Igor Loi, Antonio Pullini, Florent Rotenberg, and Luca Benini. 2018. GAP-8: A RISC-V SoC for AI at the Edge of the IoT. In *Proc. of IEEE ASAP*. 1–4.

[18] Daniel Friesel, Lennart Kaiser, and Olaf Spinczyk. 2021. Automatic Energy Model Generation with MSP430 EnergyTrace. In *Proc. of CPS-IoTBench*. 26–31.

[19] Graham Gobieski, Nathan Beckmann, and Brandon Lucia. 2019. Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems. In *Proc. of ACM ASPLOS*. 199–213.

[20] Chengyue Gong, Zixuan Jiang, Dilin Wang, Yibo Lin, Qiang Liu, and David Z Pan. 2019. Mixed Precision Neural Architecture Search for Energy Efficient Deep Learning. In *Proc. of. IEEE/ACM ICCAD*. 1–7.

[21] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proc. of ACM/IEEE ISCA*. 243–254.

[22] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. 2019. FPGA/DNN Co-Design: An Efficient Design Methodology for IoT Intelligence on the Edge. In *Proc. of ACM/IEEE DAC*. 1–6.

[23] Kaiming He and Jian Sun. 2015. Convolutional Neural Networks at Constrained Time Cost. In *Proc. of IEEE CVPR*. 5353–5360.

[24] Josiah Hester and Jacob Sorber. 2017. New Directions: The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proc. of ACM SenSys*. 1–6.

[25] Texas Instruments. 2014. EnergyTrace. http://www.ti.com/tool/ENERGYTRACE.

[26] Texas Instruments. 2016. Low-energy Accelerator (LEA). http://www.ti.com/lit/an/slaa720/slaa720.pdf.

[27] Texas Instruments. 2018. MSP430FR5994 MCU. http://www.ti.com/product/MSP430FR5994.

[28] Maxim Integrated. 2021. MAX78000 Ultra-Low-Power MCU with Arm Cortex-M4 and a Convolutional Neural Network Accelerator. https://datasheets.maximintegrated.com/en/ds/MAX78000.pdf.

[29] Bashima Islam and Shahriar Nirjon. 2020. Scheduling Computational and Energy Harvesting Tasks in Deadline-Aware Intermittent Systems. In *Proc. of IEEE RTAS*. 95–109.

[30] Bashima Islam and Shahriar Nirjon. 2020. Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems. *Proc. of ACM IMWUT/UBICOMP* 4, 3, Article 82 (2020), 29 pages.

[31] Hrishikesh Jayakumar, Arnab Raha, Jacob R. Stevens, and Vijay Raghunathan. 2017. Energy-Aware Memory Mapping for Hybrid FRAM-SRAM MCUs in Intermittently-Powered IoT Devices. *ACM TECS* 16, 3 (2017), 65:1–65:23.

[32] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. 2020. Standing on the Shoulders of Giants: Hardware and Neural Architecture Co-search with Hot Start. *IEEE TCAD* 39, 11 (2020), 4154–4165.

[33] Weiwen Jiang, Lei Yang, Edwin Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. 2020. Hardware/Software Co-Exploration of Neural Architectures. *IEEE TCAD* 39, 12 (2020), 4805–4815.

[34] Weiwen Jiang, Xinyi Zhang, Edwin H.-M. Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. 2019. Accuracy vs. Efficiency: Achieving Both through FPGA-Implementation Aware Neural Architecture Search. In *Proc. of ACM/IEEE DAC*. 1–6.

[35] Chih-Kai Kang, Chun-Han Lin, Pi-Cheng Hsiu, and Ming-Syan Chen. 2018. HomeRun: HW/SW Co-Design for Program Atomicity on Self-Powered Intermittent Systems. In *Proc. of ACM/IEEE ISLPED*. 29:1–29:6.

[36] Chih-Kai Kang, Hashan Roshantha Mendis, Chun-Han Lin, Ming-Syan Chen, and Pi-Cheng Hsiu. 2020. Everything Leaves Footprints: Hardware Accelerated Intermittent Deep Inference. *IEEE TCAD* 39, 11 (2020), 3479–3491.

[37] Kendryte. 2018. K210 AI Chip Datasheet. https://s3.cn-north-1.amazonaws.com.cn/dl.kendryte.com/documents/kendryte_datasheet_20181011163248_en.pdf.

[38] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images.* Technical Report. University of Toronto.

[39] Seulki Lee and Shahriar Nirjon. 2019. Neuro.ZERO: a Zero-Energy Neural Network Accelerator for Embedded Sensing and Inference Systems. In *Proc. of ACM SenSys*. 138–152.

[40] Jiajun Li, Guihai Yan, Wenyan Lu, Shuhao Jiang, Shijun Gong, Jingya Wu, and Xiaowei Li. 2018. SmartShuttle: Optimizing Off-Chip Memory Accesses for Deep Learning Accelerators. In *Proc. of DATE*. 343–348.

[41] Yuhong Li, Cong Hao, Xiaofan Zhang, Xinheng Liu, Yao Chen, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2020. EDD: Efficient Differentiable DNN Architecture and Implementation Co-search for Embedded AI Solutions. In *Proc. of ACM/IEEE DAC*. IEEE, 1–6.

[42] Yuyang Li, Yawen Wu, Xincheng Zhang, Ehab Hamed, Jingtong Hu, and Inhee Lee. 2021. Developing a Miniature Energy-Harvesting-Powered Edge Device with Multi-Exit Neural Network. In *Proc. IEEE ISCAS*. 1–5.

[43] Yun Liang, Liqiang Lu, Yicheng Jin, Jiaming Xie, Ruirui Huang, Jiansong Zhang, and Wei Lin. 2021. An Efficient Hardware Design for Accelerating Sparse CNNs with NAS-based Models. *IEEE TCAD (Early access)* (2021), 1–1.

[44] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny Deep Learning on IoT Devices. In *Proc. of NeurIPS*.

[45] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018. Hierarchical Representations for Efficient Architecture Search. In *Proc. of ICLR*.
[46] Qing Lu, Weiwen Jiang, Xiaowei Xu, Yiyu Shi, and Jingtong Hu. 2019. On Neural Architecture Search for Resource-Constrained Hardware Platforms. In *Proc. of ICCAD*. 1–8.
[47] Kaisheng Ma, Xueqing Li, Shuangchen Li, Yongpan Liu, John Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Nonvolatile Processor Architecture Exploration for Energy-Harvesting Applications. *IEEE Micro* 35, 5 (2015), 32–40.
[48] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Architecture Exploration for Ambient Energy Harvesting Nonvolatile Processors. In *Proc. of IEEE HPCA*. 526–537.
[49] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2018. Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA. *IEEE TVLSI* 26, 7 (Jul 2018), 1354–1367.
[50] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent Execution Without Checkpoints. In *Proc. of ACM OOPSLA*. 96:1–96:30.
[51] Paolo Meloni, Daniela Loi, Paola Busia, Gianfranco Deriu, Andy D. Pimentel, Dolly Sapra, Todor Stefanov, Svetlana Minakova, Francesco Conti, Luca Benini, Maura Pintor, Battista Biggio, Bernhard Moser, Natalia Shepeleva, Nikos Fragoulis, Ilias Theodorakopoulos, Michael Masin, and Francesca Palumbo. 2019. Optimization and Deployment of CNNs at the Edge: The ALOHA Experience. In *Proc. of ACM CF*. 326–332.
[52] Hashan Roshantha Mendis, Chih-Kai Kang, and Pi-Cheng Hsiu. 2021. iNAS Open Source Project. https://github.com/EMCLab-Sinica/Intermittent-aware-NAS.
[53] Andy D Pimentel, Mark Thompson, Simon Polstra, and Cagkan Erbas. 2008. Calibration of Abstract Performance Models for System-Level Design Space Exploration. *Journal of Sig. Proc. Sys.* 50, 2 (2008), 99–114.
[54] Antonio Polino, Razvan Pascanu, and Dan-Adrian Alistarh. 2018. Model compression via distillation and quantization. In *6th International Conference on Learning Representations*.
[55] Dolly Sapra and Andy D Pimentel. 2020. An Evolutionary Optimization Algorithm for Gradually Saturating Objective Functions. In *Proc. of GECCO*. 886–893.
[56] Yongming Shen, Michael Ferdman, and Peter Milder. 2017. Maximizing CNN Accelerator Efficiency through Resource Partitioning. In *Proc. of ACM/IEEE ISCA*. 535–547.
[57] Sivert T. Sliper, Domenico Balsamo, Alex S. Weddell, and Geoff V. Merrett. 2018. Enabling Intermittent Computing on High-performance Out-of-order Processors. In *Proc. of ACM ENSsys (ENSsys '18)*. 19–25.
[58] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.
[59] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *Proc. of IEEE CVPR*. 2820–2828.
[60] Sumanth Umesh and Sparsh Mittal. 2020. A Survey of Techniques for Intermittent Computing. *Journal of Sys. Arch* 112 (Aug 2020), 101859.
[61] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. 2020. APQ: Joint Search for Network Architecture, Pruning and Quantization Policy. In *Proc. of IEEE CVPR*. 2078–2087.
[62] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv preprint arXiv:1804.03209* (2018).
[63] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. 2020. Neural Predictor for Neural Architecture Search. In *Proc. of ECCV*. 660–676.
[64] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. 2008. The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools. *ACM TECS* 7, 3, Article 36 (May 2008), 53 pages.
[65] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *Proc. of the IEEE CVPR*. 10734–10742.
[66] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. 2020. Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks. In *Proc. of ACM/IEEE DAC*. 1–6.
[67] Wu Yawen, Wang Zhepeng, Jia Zhenge, Shi Yiyu, and Hu Jingtong. 2020. Intermittent Inference with Nonuniformly Compressed Multi-Exit Neural Network for Energy Harvesting Powered Devices. In *Proc. of ACM/IEEE DAC*. 1–6.
[68] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proc. of ACM/SIGDA FPGA*. 161–170.

[69] Xinyi Zhang, Clay Patterson, Yongpan Liu, Chengmo Yang, Chun Jason Xue, and Jingtong Hu. 2020. Low Overhead Online Data Flow Tracking for Intermittently Powered Non-Volatile FPGAs. *ACM JETC* 16, 3 (2020), 1–20.
[70] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *Proc. of ICLR*.