Internet of Things 33 (2025) 101688

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

Fast data aware neural architecture search via supernet accelerated evaluation

Emil Njor [a],*, Colby Banbury [b], Xenofon Fafoutis [a]

[a] Technical University of Denmark, Anker Engelunds Vej 101, Kongens Lyngby, 2800, Denmark
[b] Microsoft, 1 Memorial Dr, Cambridge, MA, 02142, United States of America

ARTICLE INFO

MSC:
68T10
68T20
68T45

Keywords:
TinyML
Neural architecture search
Data aware machine learning
Embedded machine learning

ABSTRACT

Tiny machine learning (TinyML) promises to revolutionize fields such as healthcare, environmental monitoring, and industrial maintenance by running machine learning models on low-power embedded systems. However, the complex optimizations required for successful TinyML deployment continue to impede its widespread adoption.

A promising route to simplifying TinyML is through automatic machine learning (AutoML), which can distill elaborate optimization workflows into accessible key decisions. Notably, Hardware Aware Neural Architecture Searches — where a computer searches for an optimal TinyML model based on predictive performance and hardware metrics — have gained significant traction, producing some of today's most widely used TinyML models.

TinyML systems operate under extremely tight resource constraints, such as a few kB of memory and an energy consumption in the mW range. In this tight design space, the choice of input data configuration offers an attractive accuracy-latency tradeoff. Achieving truly optimal TinyML systems thus requires jointly tuning both input data and model architecture.

Despite its importance, this "Data Aware Neural Architecture Search" remains underexplored. To address this gap, we propose a new state-of-the-art Data Aware Neural Architecture Search technique and demonstrate its effectiveness on the novel TinyML "Wake Vision" dataset. Our experiments show that across varying time and hardware constraints, Data Aware Neural Architecture Search consistently discovers superior TinyML systems compared to purely architecture-focused methods, underscoring the critical role of data-aware optimization in advancing TinyML.

1. Introduction

Today, low-power microcontrollers are ubiquitous, embedded within Internet of Things (IoT) systems where they perform specialized tasks such as controlling appliances and monitoring sensors [1]. By integrating intelligent Machine Learning (ML) systems into these microcontrollers, we can enable them to make intelligent decisions and adapt to changing conditions without a wireless connection. This empowers devices to anticipate needs, simplify interactions, and ultimately improve quality of life. This vision is at the core of the TinyML field, which has already demonstrated significant impact across diverse applications such as environmental monitoring [2], predictive maintenance [3], and healthcare [4].

Successfully deploying ML systems, particularly deep learning systems which are inherently resource-intensive, on microcontrollers is challenging. These low-power devices' strict limitations on computing, power, and memory are significantly more

* Corresponding author.
E-mail address: emjn@dtu.dk (E. Njor).

https://doi.org/10.1016/j.iot.2025.101688
Received 10 February 2025; Received in revised form 31 May 2025; Accepted 24 June 2025
Available online 5 July 2025

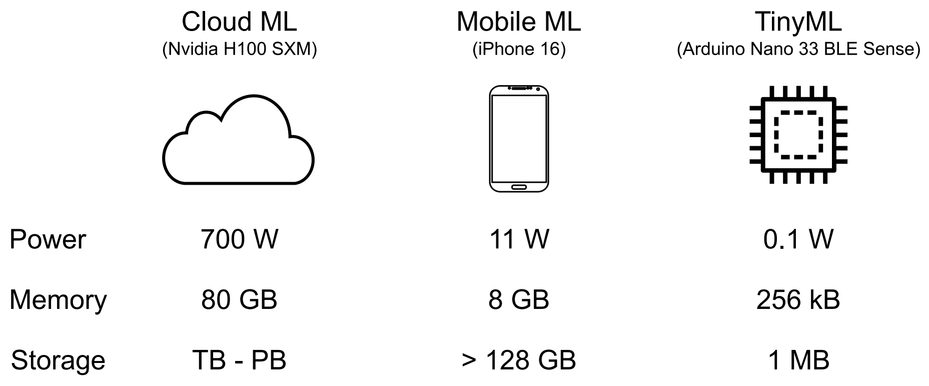

Fig. 1. An illustrative example of the resources available in cloud, mobile, and TinyML.
Sources: [5–8].

constraining than any other contemporary computing platform as visualized in Fig. 1. ML at this scale therefore demand substantial optimizations to produce feasible systems. Such optimizations include tailoring the hardware platform, inference library, ML model, and data pre-processing pipeline to meet application-specific resource constraints — a process that, with current tools, demands significant Non-Recurrent Engineering (NRE) effort.

This challenge is further exacerbated by the heterogeneous landscape of microcontrollers, which consist of a wide variety of models and manufacturers. While individual microcontrollers are inexpensive, replacing a large fleet of deployed devices can be prohibitively costly or impractical, especially for systems deployed in remote and hard-to-reach locations. Consequently, TinyML systems in reality often operate on heterogeneous hardware platforms, each requiring customized NRE work to fit varying resource capabilities.

The high NRE costs associated with deploying TinyML systems create a significant barrier to adoption. To address this, AutoML techniques have emerged as a promising solution for automating the optimization process. In particular, Hardware Aware NAS [9] is regarded as a promising research direction to scale the process of creating lean TinyML systems.

Alongside scaling the creation of TinyML systems, the speed at which NAS evaluates Neural Network (NN) architectures has also proven useful for creating higher-performing architectures than manually designed architectures. Models such as MCUNet [10] and Micronets [6], both designed through Hardware-Aware NAS, are widely recognized benchmarks for resource-efficient TinyML systems.

While Hardware-Aware NAS optimizes the resource usage of neural networks, it considers only the model architecture and overlooks the role of input data configurations. Adapting input data configurations can be useful as datasets tend to contain redundancy as information gathering is overprovisioned to ensure enough data. Because of this redundancy, it is often possible to use a reduced configuration of a dataset, e.g., by reducing image resolution or sample rate, while still being able to meet application requirements and thereby reducing resource consumption.

To bridge this gap, researchers have recently introduced "Data-Aware NAS" [11], an approach that extends Hardware-Aware NAS by simultaneously searching for the optimal input data configuration alongside the model architecture [11]. Using Data-Aware NAS, the researchers create TinyML systems that are as performant as those created using Hardware-Aware NAS, while consuming fewer system resources.

In this work, we continue this thrust towards Data Aware NAS by showing that this extension offers two key advantages:

Improved resource efficiency. Data Aware NAS can reduce data granularity, leading to smaller input data, intermediate NN representations, and model parameters. This reduction can minimize resource usage while still enabling the system to meet predictive performance constraints.

Enhanced predictive performance. Under strict resource constraints, Data-Aware NAS can identify optimal trade-offs between data granularity and model complexity, resulting in improved predictive performance compared to traditional Hardware-Aware NAS approaches.

Expanding the NAS search space to include input data configurations has the disadvantage of increasing the search space size manyfold. Although this could have introduced the challenge of maintaining tractable search times, our experiments show that a Data Aware NAS outperforms a traditional Hardware Aware NAS across varying time constraints.

In their initial exploration of Data Aware NAS [11], the authors developed a simple layer-based approach and validated its effectiveness on the ToyADMOS dataset [12]. Although this work successfully demonstrated the Data Aware NAS concept, the simplicity of the ToyADMOS dataset made it an unrealistically easy task to solve, limiting the ability to show Data Aware NAS working under more realistic and challenging conditions.

In this paper, we advance the state of Data Aware NAS by applying it to the significantly more complex Wake Vision dataset containing around 6 million person/no-person images in diverse and realistic environments [13]. Exploring TinyML systems for using this dataset would be infeasible using the prior Data Aware NAS implementation due to speed limitations. Therefore, to overcome this barrier, we integrate state-of-the-art Supernet-based NAS techniques in a new Data Aware NAS implementation to ensure tractable search times. Specifically, we make the following contributions:

Improved performance under resource constraints. We demonstrate that Data-Aware NAS consistently outperforms traditional Hardware-Aware NAS in creating TinyML systems that achieve better predictive performance within the same resource constraints.

Efficient search times. We show that Data-Aware NAS discovers superior TinyML systems compared to purely architecture-focused methods regardless of time constraints.

Reduced expert intervention. Compared to Hardware-Aware NAS, our approach reduces the expert knowledge required to design and create efficient TinyML systems.

Supernet-based evaluation. We contribute an improved Data Aware NAS based on supernets, significantly accelerating the NAS evaluation process and enabling the discovery of optimized systems for complex datasets and search spaces.

The remaining sections of the paper are organized as follows:

Section 2: Related work. Presents prior scientific works that closely align with the challenges and solutions presented in this work.

Section 3: Resource consumption analysis. Examines the typical resource constraints encountered in TinyML and analyzes the resource interplay between data and TinyML models.

Section 4: Improving data aware neural architecture search. Introduces the enhanced Data Aware NAS implementation enabling the experiments presented in this work.

Section 5: Results. Presents the experimental outcomes of our enhanced Data Aware NAS approach and discusses the insights drawn from them.

Section 6: Future work. Identifies promising directions for extending this research.

Section 7: Conclusion. Highlights key findings and contributions of the work.

Acknowledgments. Recognizes the funding sources that enabled this research.

Declaration of generative AI and AI-assisted technologies in the writing process. Discloses the use of generative Artificial Intelligence (AI) for refining the manuscript's language.

2. Related work

The research presented in this paper intersects multiple fields. First, it is based on the idea of using a computer algorithm to optimize NN architectures, commonly known as NAS. Second, it considers the deployment of resource-intensive NN models on severely resource-constrained embedded devices, a domain known as TinyML. This has so far been achieved either by handcrafting manual NN architectures or via Hardware Aware NASs. Third, it intersects the growing trend of Data-Centric AI.

2.1. Neural architecture search

NAS is a widely explored approach for designing high-performance NN architectures in traditional computational scales [14–17]. Early NAS systems relied on evolutionary algorithms [18] and reinforcement learning [14,15] to explore discrete search spaces of potential NN architectures.

Recently, supernet-based methods, which train a single large model and extract high-quality subarchitectures from it, have gained popularity as a way to reduce search times [17]. Differentiable Neural Architecture Search (DNAS) has also emerged as a promising technique, where a discrete NN search space is turned continuous through proxy scalar values, enabling fast searches through gradient-based optimization methods [19].

2.2. TinyML neural architecture search

The application of NAS has been central in creating models that balance performance and efficiency to run on TinyML hardware since the very beginning of the field. SpArSe [20], one of the first works to show that Convolutional Neural Networks (CNNs) can be deployed on TinyML hardware, uses Bayesian-based NAS to find suitable CNN architectures.

More recent works on TinyML NAS include MCUNet [10] and Micronets [6], used to create the MCUNet and Micronet family of models, which are among the most used in TinyML.

The MCUNet NAS consists of two steps. During the first step, a search space is created where most models fit tightly into the resource constraints of a TinyML device. In the second step, a one-shot NAS is performed on the search space, after which an evolutionary search is used to sample the best sub-network. The MCUNet NAS arguably includes data in its search space as the adapted search space for TinyML is created by, among others, reducing the input resolution of the search space [10].

The authors of the Micronets paper use DNAS [19] based on either a MobileNetv2 [21] or a DS-CNN(L) [22] backbone to design TinyML models for Person Detection, Keyword Spotting, and Anomaly Detection.

Other works on NAS for TinyML include CNAS, which optimizes NN for hardware device operator constraints [23] and μNAS, which proposes a NAS with a highly granular search space [24].

2.3. Manually designed TinyML architectures

While NAS dominates TinyML model development, several works propose manually designed architectures. For instance, CoopNet is a manually designed model that utilizes heterogeneous quantization and binarization to produce an efficient model architecture [25]. Another example is the DS-CNN model, which uses depthwise separable convolution layers to achieve superior performance in keyword spotting applications within TinyML resource constraints [22].

2.4. Data-centric artificial intelligence

An up-and-coming research area that has traditionally received little attention is Data-Centric AI. Where many research papers propose techniques for optimizing ML models, much fewer papers consider ways to improve ML through data-centric innovations [26]. Techniques such as weak supervision for fast dataset labeling [27], confident learning for automated label error spotting and correction [28], and active learning where only the most important data samples are labeled [29] are key contributions to this domain.

2.5. Data aware neural architecture search

Researchers recently introduced Data Aware NAS [11], which incorporates data configurations into the search space of a Hardware Aware NAS. Using evolutionary algorithms, they jointly optimized data configurations and CNN models within a layer-based search space. Their findings demonstrate that Data Aware NAS can significantly reduce resource consumption in TinyML systems.

Building on this foundation, this present study conducts a comprehensive comparison between Data Aware- and traditional Hardware-Aware NAS through experiments on a more complex dataset. This comparison would not be possible using the prior Data Aware NAS implementation due to speed limitations. To overcome this barrier, we contribute a new Data Aware NAS implementation, built on state-of-the-art NAS techniques.

3. Resource availability and consumption in TinyML systems

Given the scarce amount of resources available for a TinyML system, it is crucial to accurately track its resource consumption to make the most of the available resources. To properly evaluate the resource consumption of TinyML systems, we first need to establish the resources available in such systems and how the components of a TinyML system consume these. We do so in this section by first characterizing the available resources in TinyML systems, then examining how data pre-processing impacts resource usage, and finally analyzing the resource consumption of running TinyML models.

3.1. Available resources

The available resources in a TinyML system are primarily decided by the hardware platform on which it is deployed. Typical resources that we consider are computational capacity, volatile RAM, persistent flash storage, and energy [6,10]. These resources are interdependent, with energy consumption in particular being tightly coupled to the usage of other resources.

Consider the Arduino Nano 33 BLE Sense [8], a popular device in the TinyML community. It features an ARM Cortex M4 processor supporting Single Instruction Multiple Data (SIMD) parallel processing for operations like Multiply-Accumulates (MACs), commonly used in NNs. The device provides 256 kB of Static RAM (SRAM) and 1 MB of flash storage. While there are no official figures for the energy consumption of the device, our measurements show a power draw of around 125 mW during ML inference.

3.2. Input data resource consumption

In a typical TinyML system, input data undergoes several steps. First, it is captured by a sensor, then transferred to the Microcontroller Unit (MCU) memory, pre-processed, and lastly fed to the TinyML model. Quickly executing these steps requires fast and efficient data transfer. Furthermore, as data usually will not need to be stored after being fed to a TinyML model, there is typically no need for memory to be able to persist data across power cycles. Among the memory types typically present in a typical MCUs, RAM fits these characteristics best and is therefore typically used as working memory. The data pre-processing step additionally relies on computational resources to process data. By adapting input data configurations to lower granularities we can decrease the processing and memory resources required for moving and processing input data.

Once input data has been preprocessed it is ready to be fed to the TinyML model. This usually happens in one of two ways:

Streaming Data is processed one sample at a time as it is captured. Also known as online processing.

Batched Data is accumulated in a buffer until enough samples are collected for parallel processing. Also known as offline processing.

The batch paradigm consumes more memory as sensor data needs to be aggregated and preserved in memory for longer than with streaming. However, batched processing can leverage improved parallel processing and, therefore, improve the inference efficiency of the model. Furthermore, batching of sensor data is required to predict over the temporal dimension (e.g., audio data), except in some rare instances where recurrent neural networks are used. Due to the tight memory constraints of TinyML systems, large batches are typically infeasible, and streaming is used more frequently.

Even when input is passed to a model, its size can have a major impact on the overall resource consumption of the model. We will cover this resource consumption in Section 3.3.

3.3. TinyML model resource consumption

Most embedded applications statically allocate memory and storage, which enables us to accurately predict the consumption of each by the architecture of the deployed NN.

For storage (i.e., non-volatile flash) consumption, the primary contributor is the number of parameters in the model and their data type (e.g., int8, fp16, etc.). As shown in Eq. (1), one can sum the number of parameters in a model, multiply by the size of each parameter, which is determined by the datatype, and factor in some overhead for the rest of the application stack [6]. In some NN layers, for example, dense layers, the number of parameters scales with the size of the input data. Hence, the choice of data granularity can affect the storage consumption of a TinyML model.

An approximate working memory (i.e., RAM) consumption of a model can be computed using Eq. (2). Informally, this equation finds the maximum sum of the input and output intermediate tensors over every layer in a network, factoring in any residuals or other tensors that need to be preserved in memory. The process of calculating the size of input and output intermediate tensors differs for each layer and can be dependent on both the neural architecture and the input data — hence again the choice of data granularity can affect the working memory consumption of a TinyML model.

$$
\begin{aligned}
&c_f = \sum m_p t_s \\
&\quad \text{where} \quad c_f \quad : \text{Flash Consumption in Bytes,} \\
&m_p \quad : \text{Model Parameter,} \\
&t_s \quad : \text{Size of Parameter Datatype in Bytes.}
\end{aligned} \tag{1}
$$

$$
\begin{aligned}
&c_r = \max\left(d + l_{1o} \in m, \max_{l_j \in m}\left(l_{ji} + l_{jo}\right)\right) \\
&\quad \text{where} \quad c_r \quad : \text{RAM Consumption in Bytes,} \\
&d \quad : \text{Size of Input Data in Bytes,} \\
&l_j \quad : \text{A layer in a Model,} \\
&m \quad : \text{The set of all layers in a Model,} \\
&l_{1o} \quad : \text{Size of First Model Layer Output in Bytes,} \\
&l_{ji} \quad : \text{Size of Layer } j \text{ Input in Bytes,} \\
&l_{jo} \quad : \text{Size of Layer } j \text{ Output in Bytes}
\end{aligned} \tag{2}
$$

Computational resources are not “consumed” in the same way, but they determine if a given model is computed within the latency budget of the application. To predict latency, you can often use proxy metrics, such as the number of operations, or predictive surrogate models [9]. To get the best accuracy, one must measure the real latency on the device itself. The same goes for energy consumption, which, given the relatively simple hardware architecture of MCUs, is inextricably linked to the latency.

4. Improving data aware neural architecture search

While prior work demonstrated that a Data Aware NAS could reduce resource consumption of TinyML systems under predictive performance constraints, it failed to show the trading-off of data and model resources to improve predictive performance [11].

To show this phenomenon in this paper, we must target a less trivial application than previously [11]. A promising application in TinyML is Person Detection, where an image is to be classified according to whether it contains a person [30]. As this is a vision application, it will work on higher dimensionality data than previously considered in Data Aware NAS, enabling a deeper exploration of resource-performance trade-offs.

Targeting Person Detection allows us to utilize the new “Wake Vision” dataset [13], recently introduced to the TinyML domain, promising vast improvements in size and quality over traditional TinyML datasets.

4.1. A supernet based data aware neural architecture search

The increased complexity of Person Detection, coupled with the larger Wake Vision dataset, makes the prior approach of using a layer-based search space and naive evolutionary algorithms too slow for practical use. To address this, we propose a new supernet-based implementation utilizing a MobileNetv2 [21] backbone. MobileNetv2 is a lightweight CNN model designed for Mobile computing, thereby being large enough to encompass a good amount of embedded-scale subnetworks.

In this approach, a fully sized MobileNetv2 supernet is trained on an Nvidia V100 system for each input data configuration. Training is performed lazily, meaning it is initiated only right before the supernet’s first use. During the evaluation, the Data Aware NAS samples a model configuration from the supernet corresponding to its data configuration, which provides a model with pre-trained weights. This model is then further fine-tuned to learn weights appropriate for its neural architecture. This fine-tuning is significantly faster than training a model from scratch, thereby enabling fast evaluation.

Eqs. (3) and (4) illustrates the difference in search time between traditional NAS and supernet-based NAS.

$$\begin{aligned} &30.000\ F + 100\ N \\ &\text{where}\quad F = \text{A factor of the Supernet size} \\ &\qquad\qquad\quad \text{compared to extracted models,} \\ &\qquad\quad\ N = \text{Number of generated models} \end{aligned} \tag{3}$$

$$\begin{aligned} &25.000\ N \\ &\text{where}\quad N = \text{Number of generated models} \end{aligned} \tag{4}$$

We evaluate each TinyML system using the fitness function described in Appendix B and use a tournament-based genetic algorithm to produce new candidate TinyML systems based on previous ones. The final output of the system is a collection of Pareto optimal TinyML systems, i.e., systems that are each superior to every other generated TinyML system in some way.

Input data configuration options. We configure the new Data Aware NAS implementation to consider two orthogonal ways to configure its data. These configuration options are based on engineering knowledge and provide no guarantees of containing the optimal value for each parameter. To the best of our knowledge, there are no standardized search spaces that we can use, but the search spaces we define closely follow other works on NAS for TinyML [10].

First, we allow it to configure the resolution of the image that is passed onto the NN model. Possible resolution options span:

$$R = \{32, 64, 96, 128, 160, 192, 224\}.$$

Large input resolutions profoundly impact the RAM consumption of data due to the increased shape of the tensor storing the image. It also has a significant effect on the size, and therefore RAM consumption, of the internal representations, i.e., activations, of the NN model created by the convolutional and depthwise separable convolutional layers, both of which are frequent in a MobileNetV2 model. The number of parameters of a model is less affected by input resolution changes, as parameters in convolutional and depthwise separable convolutional layers depend on fixed filter sizes, not tensor dimensions.

Secondly, we enable the Data Aware NAS to adapt whether the image is encoded in monochrome or Red, Green, and Blue (RBG) format, i.e., $C = \{\text{monochrome}, \text{rbg}\}$. If a monochrome image encoding is picked, we reduce the width (also called alpha) [21] of the MobileNetv2 supernet by $1/3$. Consequently, this change affects both the RAM and flash consumption of the final TinyML system.

Model configuration options. We consider two ways to adapt the MobileNetv2 architecture. As with the input search space, this search space is also defined using the engineering knowledge of the authors. The first is to adapt the depth of the network. A MobileNetv2 architecture is made up of inverted bottleneck blocks [21] that, over time, reduce the resolution of the input image while greatly increasing its channels (also often called features). These blocks are divided into seven overall stages, see Fig. 2, inside which the internal representation shape between the blocks is kept constant. Because of this structure, it is possible to remove blocks from these stages as long as one block in each stage remains to update the internal representation shape for the next stage. We keep the initial two MobileNetV2 blocks intact to enable them to extract low-level features for later layers. With that in mind, we define our model depth search space as described in Eq. (5).

$$\begin{aligned} &(b_3, b_4, b_5, b_6, b_7) \in B_3 \times B_4 \times B_5 \times B_6 \times B_7 \\ &\qquad \text{where}\quad B_3 = \{0, 1, 2, 3\} \\ &\qquad\qquad\qquad B_4 = \{0, 1, 2, 3, 4\} \\ &\qquad\qquad\qquad B_5 = \{0, 1, 2, 3\} \\ &\qquad\qquad\qquad B_6 = \{0, 1, 2, 3\} \\ &\qquad\qquad\qquad B_7 = \{0, 1\} \\ &\qquad\qquad\qquad\ b_i \in B_i \end{aligned} \tag{5}$$

We also enable the possibility of completely removing stages if downstream stages are also removed. For example, it is possible to remove stage 6, given that stage 7 is also removed by attaching the final classification head to the end of stage 5. Our search strategy ensures that infeasible architectures containing blocks after a completely removed stage are not generated during the search and that this does not adversely affect the probability of finding deeper architectures. Adapting the depth of the network has no implications on the RAM requirements to store data but can, depending on the stage and block, have a large impact on the RAM consumption of internal model representations and persistent memory taken up by the total number of parameters in the model.

The second approach to adapting the MobileNetv2 architecture is to change the width of the network, that is, the number of channels created by the convolutional layers of the model. The width of the network is controlled by a single alpha parameter $A = \{0.1, 0.2, \dots, 1\}$. For implementation reasons, this is achieved using masking layers as introduced in FBNetV2 [31]. This change significantly impacts both the size of internal model representations and the total number of parameters in the model.

Due to the complexity of modeling model latency and energy consumption accurately, our implementation limits its hardware-aware metrics to RAM memory and flash storage. To calculate the total memory consumption of the TinyML systems, we assume a streaming data paradigm where only a single data sample is stored in memory at a time. Our implementation supports storing the data or model in datatypes of various sizes; however, all experiments in this paper are conducted assuming 8-bit datatypes for storing both data, internal model representations, and model parameters.

A URL to an open-source repository containing the new Data Aware NAS implementation can be found in Appendix A.

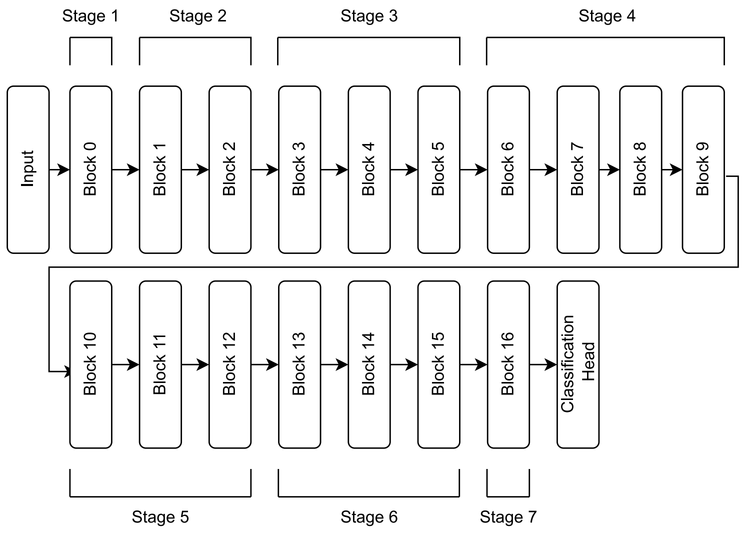

Fig. 2. The structure of a MobileNetV2 Model. Each block consists of an inverted bottleneck.

5. Results

This section presents a comprehensive evaluation of the Data Aware NAS system described in Section 4 through four complementary experiments. All experiments are conducted on a Nvidia V100 system for 23 h, with metrics such as Accuracy, Precision, and Recall all estimated on full 32 bit floating point models.

Before presenting the results of these four experiments, we first revisit results from prior work on Data Aware NAS [11] to present a self-contained and holistic evaluation of Data Aware NAS. Second, we show experimental results that Data Aware NAS is able to find TinyML systems with better predictive performance than Hardware Aware NASs (i.e., Non-Data Aware NAS) under the same resource constraints. Third, we present results that show a Data Aware NAS outperforming a Hardware Aware NAS at any time constraint. Fourth, we apply the Data Aware NAS across embedded systems of differing resource constraints, showing how Data Aware NAS can automatically adapt to resource constraints where Hardware Aware NAS necessitates manual engineering work. Fifth and last, we compare the search speed of the current supernet-based Data Aware NAS to the previously published Data Aware NAS implementation, showcasing how the technical improvements highlighted in this paper allow for the discovery of more and better TinyML systems.

Collectively, these experiments provide strong evidence that a Data Aware NAS produces superior TinyML models compared to traditional Hardware Aware NASs, achieving better performance across any timescale.

5.1. Resource consumption under performance constraints

To showcase the improvements done to Data Aware NAS in this work, we first revisit the main results from the previous publication on Data Aware NAS [11]. These results are not based on the Data Aware NAS algorithm described in Section 4, but on the simpler ToyADMOS dataset [12] and a more naive approach to Data Aware NAS. This earlier method utilized evolutionary search over a layer-based search space. For more details, we refer to the previous paper [11].

In this prior work, the authors used both a Data Aware NAS and a traditional Hardware Aware NAS to search for TinyML systems that could effectively distinguish between the normal and anomalous samples of the ToyADMOS dataset. We showcase the results of this experiment in Pareto Frontier plots in Figs. 3 and 4. Note that while all points plotted represent Pareto optimal TinyML systems in each individual run, not all points are Pareto optimal across runs. For that reason, we include a line drawing the Pareto Frontier of the Data Aware- and Hardware Aware NAS to each plot.

As evident from the TinyML systems reaching an F1-Score of 1 in these plots, both the Data Aware NAS and traditional Hardware Aware NAS are able to find models that can perfectly distinguish the normal and anomalous samples in the ToyADMOS test set. However, the Data Aware NAS can do so with TinyML systems that consume more than four magnitudes less flash memory than the TinyML systems created by the traditional Hardware Aware NAS. Unfortunately, the prior work did not track advanced resource consumption metrics such as RAM, but we expect it to have been significantly smaller on the systems created by Data Aware NAS.

These results provided experimental evidence that Data Aware NAS can be used to significantly reduce the resource consumption of TinyML Systems while maintaining performance.

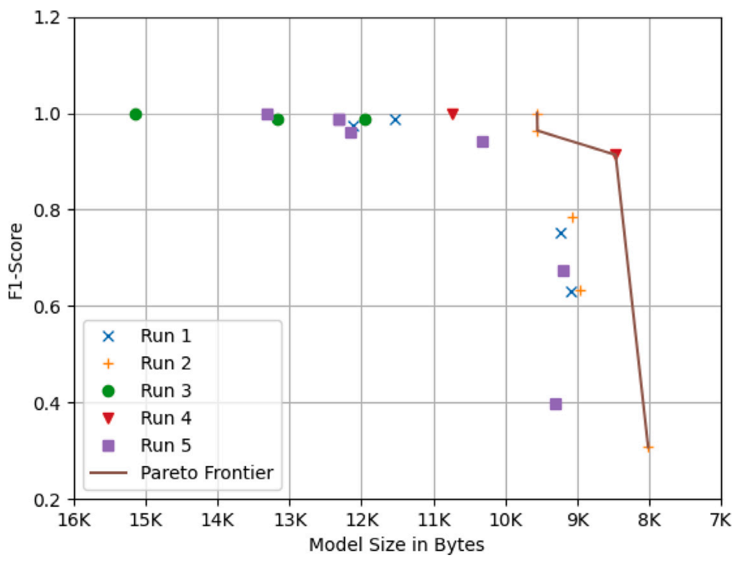

Fig. 3. Pareto Frontier of TinyML systems generated by the Data Aware NAS from [11].

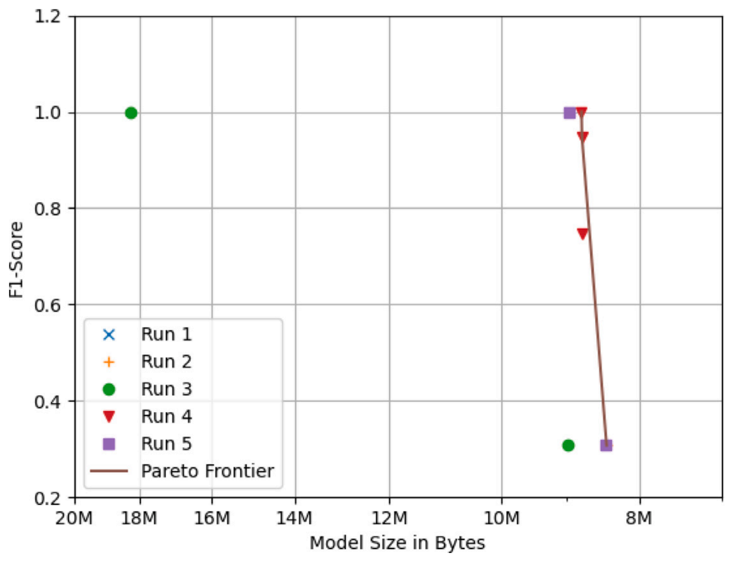

Fig. 4. Pareto Frontier of TinyML systems generated by the Hardware Aware NAS from [11].

5.2. Predictive performance under resource constraints

The first experiment provides evidence that a Data Aware NAS can reduce the overall resource requirements of a TinyML system under constraints on the system's predictive performance. A second central question is whether a Data Aware NAS can find better TinyML systems than a Hardware Aware NAS under identical resource constraints.

To investigate this, we set up an experiment where a Data Aware NAS and an equivalent Hardware Aware NAS search for a TinyML system to fit in memory constraints of 512 kB RAM and 2 MB flash memory. These memory constraints are typical for many TinyML-scale devices, for example, the STM32H743 [10] or the STMF767ZI [6].

The results of this experiment are plotted in Figs. 5 and 6 for the Data Aware- and Hardware Aware NAS respectively. These plots track the trade-off between three metrics — accuracy, RAM consumption, and flash consumption. Accuracy is tracked on the y-axis of the plot, RAM consumption on the x-axis, whereas flash consumption is tracked by the size of the individual points. The main takeaway from these figures is that the Data Aware NAS finds architectures that perform better than the Hardware Aware NAS at a maximum accuracy of 79.5% and 77.6%, respectively. Furthermore, the Data Aware NAS finds 22 models with an accuracy of above 75%, whereas the Hardware Aware NAS finds only 4 models above this threshold. The larger amount of Pareto optimal

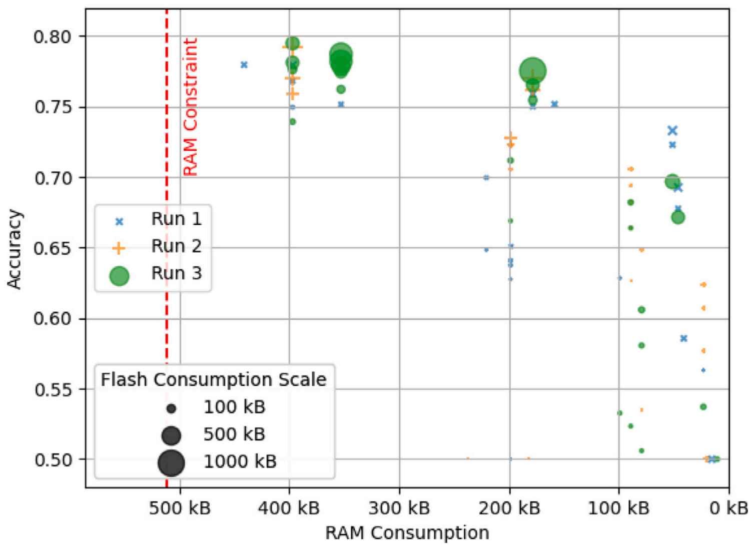

Fig. 5. Pareto optimal TinyML systems generated using a Data Aware NAS.

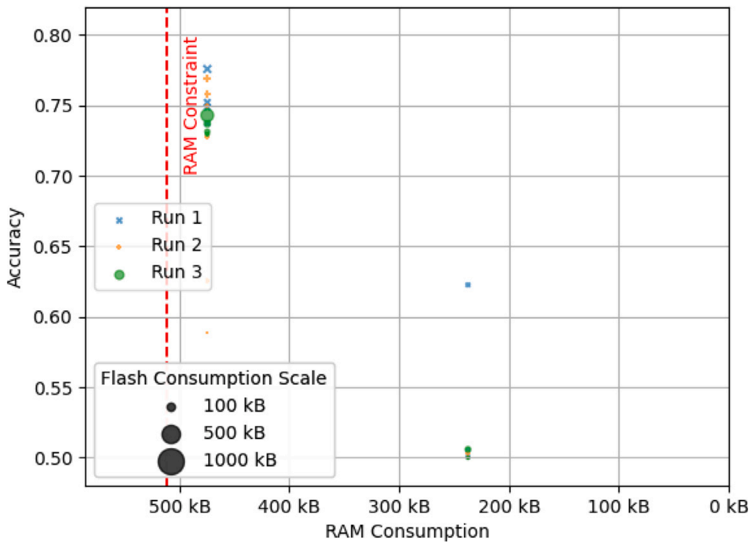

Fig. 6. Pareto optimal TinyML systems generated using a traditional Hardware Aware NAS.

models generated by Data Aware NAS would, in a practical application, mean that it would be easier to find a TinyML system fitting the requirements for an application.

These results suggest that a Data Aware NAS not only improves predictive performance but also provides more Pareto optimal choices, simplifying system selection for practical applications.

5.3. Data aware neural architecture search speed

The search space of a Data Aware NAS is a superset of the search space of a Hardware Aware NAS and is several times larger. Therefore, a relevant question is: Can a Data Aware NAS find better TinyML systems than a Hardware Aware NAS in a certain period of time?

We use the models generated throughout the two experiments from Section 5.2 to investigate this. In Fig. 7, we plot the fitness score yielded by the fitness function of our NASs (see Appendix B) for the models generated by both the Data Aware- and Hardware Aware NAS.

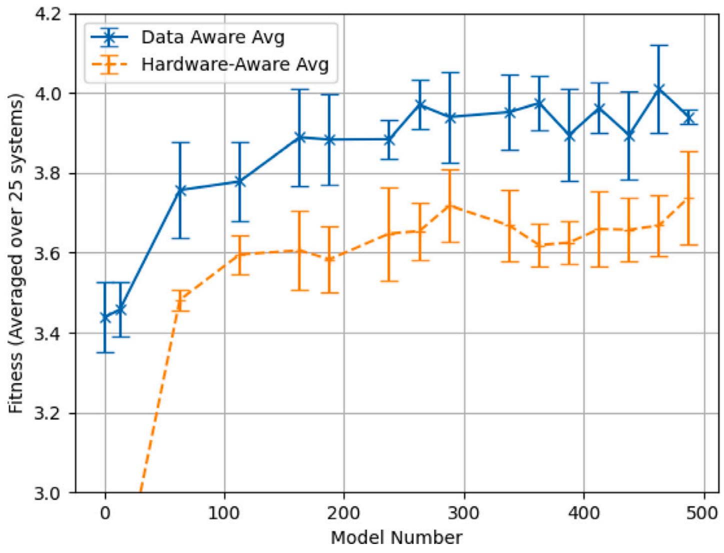

Fig. 7. Fitness evolution for both Data Aware and Hardware Aware NAS. (Fitness values averaged across 25 Models).

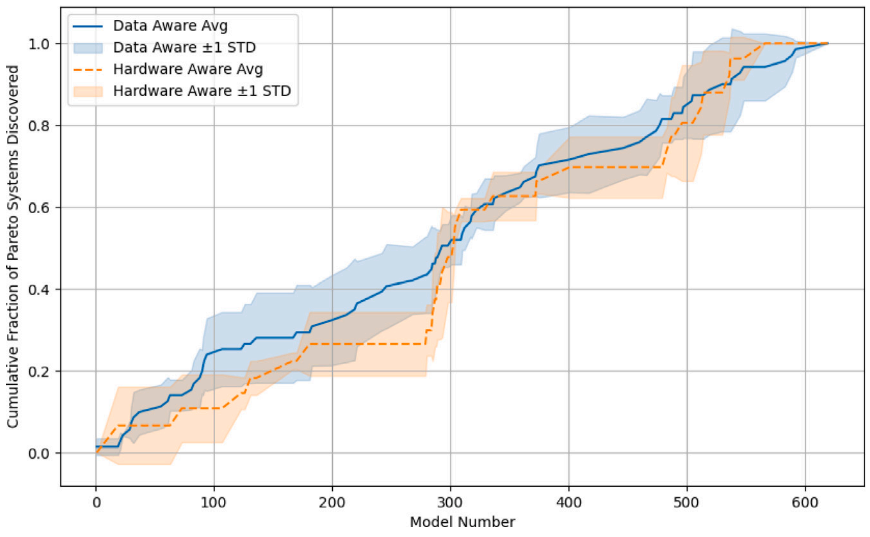

Fig. 8. Cumulative fraction of Pareto optimal models discovered throughout Data Aware and Hardware Aware searches.

Contrary to our initial hypothesis, the models generated by the Data Aware NAS outperform the models generated by the Hardware Aware NAS throughout the entire search process, which suggests that it is worth using a Data Aware NAS over a Hardware Aware NAS regardless of the search time.

Our fitness function is not a perfect proxy for when good TinyML systems are found, as systems can be Pareto optimal without achieving a high fitness score. Therefore, as a complementary analysis, we plot the cumulative fraction of Pareto optimal discovered throughout both a Data Aware- and Hardware Aware NAS in Fig. 8.

These results suggest that, while Pareto optimal systems are discovered throughout the entire search time in both NASs, a Data Aware NAS will likely find more Pareto optimal models in short searches than traditional Hardware Aware NASs.

5.4. *Ease of use across resource constraints*

The experiment in Section 5.2 shows that Data Aware NAS can create better TinyML systems than Hardware Aware NAS under the resource constraints of that experiment. A question remains: How does this generalize to other resource constraints?

To test this, we set up a Data Aware NAS to search for TinyML systems that can fit in the memory constraints of two other resource-constrained hardware platforms. For the first we select memory constraints of 320 kB and 1 MB corresponding to

Table 1
The three examples of TinyML memory constraints considered in this paper.

	RAM	Flash	Example
Large	512 kB	2 MB	STM32F765
Medium	320 kB	1 MB	STM32F746
Small	256 kB	1 MB	Arduino Nano 33 BLE Sense

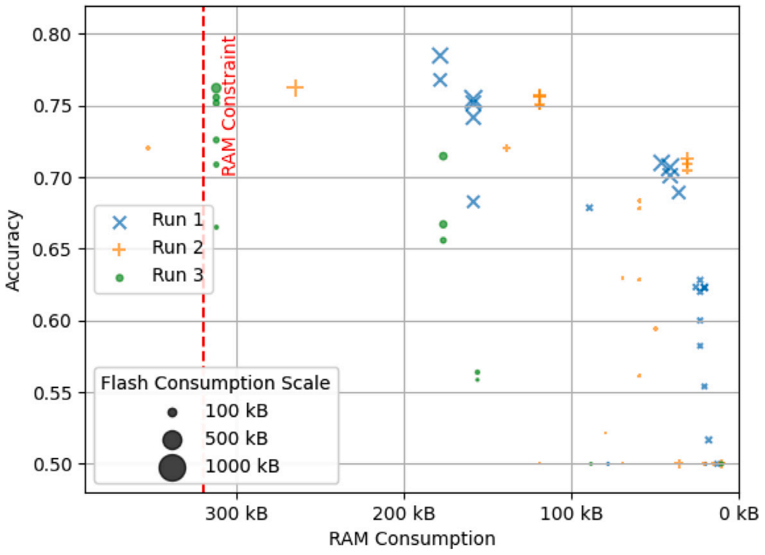

Fig. 9. TinyML systems generated by Data Aware NAS for a device with constraints of 320 kB RAM and 1 MB Flash Memory.

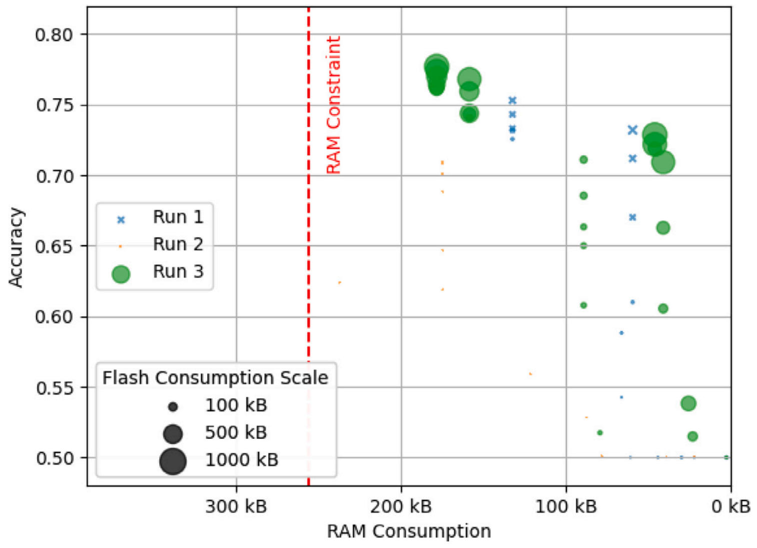

Fig. 10. TinyML systems generated by Data Aware NAS for a device with constraints of 256 kB RAM and 1 MB Flash Memory.

the STM32F746 [6,10]. As the second we set memory constraints of 256 kB and 1 MB corresponding to the STM32F412 [10], STM32F446RE [6] or the Arduino Nano 33 Bluetooth Low Energy (BLE) Sense [8]. Together, these systems form examples of a large, medium, and small TinyML hardware platform as visualized in Table 1.

The TinyML Pareto frontier of TinyML systems generated for the large system can be found in Fig. 5, while the Pareto frontiers for the medium and small devices can be found in Figs. 9 and 10 respectively.

Notice that the Pareto optimal TinyML systems for the medium-sized device contain a system from run two that exceeds the RAM constraints of that device. This is not a mistake but rather because not all constraint-violating models are excluded, only penalized by the fitness function (see Appendix B) during our NAS.

These results show that our Data Aware NAS can discover suitable TinyML across different resource constraints with a maximum accuracy of 79.5% in the large device, 78.5% in the medium device, and 77.7% for the small device. Perhaps most importantly, these results are achieved by only changing the resource constraints given to the Data Aware NAS. Thereby making a Data Aware NAS significantly easier to use than Hardware Aware NAS, which would have an engineer manually adjust many more parameters and/or data pre-processing to find TinyML systems across these resource constraints.

5.5. *Supernet speedup*

In Section 4, we claim that it is necessary to introduce a new Data Aware NAS implementation based on MobileNetV2 supernets to find good TinyML systems for Person Detection in tractable time. In this experiment, we seek to prove this by evaluating the supernet-based implementation's speedup over the original naive layer-based implementation proposed in the original work on Data Aware NAS [11].

To do so, we run a modified version of the prior naive NAS implementation for 23 h — the same time as allocated to our new supernet-based NAS in other experiments. The modifications made to the naive NAS are all done in an attempt to enable a fair apples-to-apples comparison between the two NASs. A description of the modifications done can be found in Appendix C.

Even with these modifications, creating a truly fair comparison is difficult due to the extent of changes made to improve the novel supernet-based search. For example, the search space of the two different NASs allows for discovering entirely different model architectures with varying training times and learning capacities.

Despite these caveats, after running each search for 23 h, the supernet-based NAS consistently discovers significantly better-performing systems, achieving a peak accuracy of 79.5%. In contrast, the naive NAS only finds TinyML systems reaching an accuracy of 67.7%. This shows that the supernet-based search yields TinyML systems that are approximately 11.8 percentage points more accurate than the traditional NAS search, which is ultimately what matters for ML practitioners.

6. Future work

The search for TinyML systems using Data Aware NAS has only just begun. There are still plenty of opportunities for future research into, e.g., improving Data Aware NAS, using it to create new state-of-the-art TinyML Systems, or applying it to discover theoretical insights into TinyML applications. This section identifies some of the most promising of these future research directions.

Expanded hardware metrics. Resource constraints on TinyML hardware encompass more than just the memory and computational constraints considered in this and other works on NAS for TinyML [6,10,23,32]. Inference time and energy consumption metrics are especially tough to quantify as they depend heavily on target hardware. Both are highly relevant metrics in TinyML system design. Unfortunately, these metrics are currently hard to simulate or estimate accurately without deploying TinyML systems onto real hardware — which is challenging in TinyML NAS where models must be evaluated quickly and without manual intervention. Therefore, a promising research avenue to explore is to create accurate estimator tools that can provide realistic metrics in little time and be integrated automatically into NAS tools.

Differentiable data aware neural architecture search. Many state-of-the-art NASs have recently explored using differentiable search spaces and strategies to speed up their search with promising results [19,33]. Expanding Data Aware NAS to use a differentiable search space and strategy could improve search speeds to discover even better TinyML systems. Doing so would require designing the data configuration search space in a way that makes it possible to use differentiable search methods.

TinyML domain insights using data aware neural architecture search. This paper shows how Data Aware NAS finds better-performing TinyML Person Detection systems on severely resource-constrained devices by lowering resolution and color representation. This suggests that a larger NN architecture is more important for Person Detection systems than a higher data granularity.

TinyML systems in other application domains may perform better by trading off NN architecture size for a larger data granularity. Investigating these trade-offs for different TinyML application domains may give novel insights into these domains that can prove valuable to the broader community.

Creating state of the art TinyML models using data aware neural architecture search. Although the experiments in this paper show how Data Aware NAS can produce better TinyML systems than an equivalent Hardware Aware NAS, the search space is not carefully designed to find state-of-the-art TinyML models.

Applying Data Aware NAS alongside a more carefully designed search space may be able to produce models that can outperform current state-of-the-art models such as the MCUNet models [10].

7. Conclusion

The adoption of TinyML in the industry remains slow, largely due to the deep expertise required to deploy heavily optimized TinyML systems to microcontroller scale devices. AutoML approaches such as Hardware Aware NAS are promising avenues for making otherwise heavily technical optimization techniques accessible to the broader community.

In this paper, we extend prior work on Data Aware NAS, an augmentation of Hardware Aware NAS, which considers various ways to configure input data alongside a traditional Hardware Aware NAS.

As a part of this extension, we make significant technical improvements to prior work on Data Aware NAS, moving from considering trivial datasets, search strategies, and search spaces to a state-of-the-art TinyML dataset along with a supernet-based NAS utilizing a MobileNetV2 backbone.

On the basis of these improvements, we are able to provide new results showing Data Aware NAS discovering more accurate TinyML systems than Hardware Aware NAS given the same resource constraints, doing so across any allocated NAS time and different resource constraints.

These results pave the way for using Data Aware NAS to design state-of-the-art TinyML systems and to gather new domain insights in TinyML that can be useful to all TinyML practitioners.

CRediT authorship contribution statement

Emil Njor: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Data curation, Conceptualization. **Colby Banbury:** Writing – review & editing, Writing – original draft, Methodology. **Xenofon Fafoutis:** Writing – review & editing, Supervision, Resources, Project administration, Methodology, Funding acquisition, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used Grammarly and ChatGPT in order to improve the language of the paper. After using these tools/services, the authors reviewed and edited the content as needed and takes full responsibility for the content of the published article.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Emil Njor reports financial support was provided by Innovation Fund Denmark. Emil Njor reports financial support was provided by Key Digital Technologies Joint Undertaking. Xenofon Fafoutis reports financial support was provided by Innovation Fund Denmark. Xenofon Fafoutis reports financial support was provided by Key Digital Technologies Joint Undertaking. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was jointly supported by the Digital Research Centre Denmark (DIREC) and the Distributed Artificial Intelligent Systems (DAIS) project. DIREC (https://direc.dk/) has received funding from the Innovation Fund Denmark under Grant 9142-00001B. DAIS (https://dais-project.eu/) has received funding from the ECSEL Joint Undertaking (JU) under Grant 101007273. The JU was supported in part by the European Union's Horizon 2020 research and innovation programme and Sweden, Spain, Portugal, Belgium, Germany, Slovenia, Czech Republic, Netherlands, Denmark, Norway, and Turkey. Danish participants were supported in part by the Innovation Fund Denmark under Grant 0228-00004A.

Appendix A. Implementation repository

A repository containing the Data Aware NAS implementation used in this paper can be found at: *Removed for review period to preserve anonymity*

Appendix B. Fitness function details

During both the Data Aware NAS and the traditional Hardware Aware NAS presented in this paper, TinyML systems are evaluated using a fitness function to determine how well they perform on both prediction and hardware-related metrics. The fitness function used in this work can be found in Eq. (B.1).

$$f = w_1 a + w_2 p + w_3 r + w_4 \left(1 - \frac{v_r}{x_r}\right) + w_5 \left(1 - \frac{v_f}{x_f}\right) \tag{B.1}$$

where a : Model accuracy,
p : Model precision,
r : Model recall,
v_r : RAM violation (excess usage),
x_r : Maximum allowable RAM consumption,
v_f : Flash violation (excess usage),
x_f : Maximum allowable Flash consumption,
$w_1 \cdots w_5$: Weights to adjust the importance of each term.

Maximum RAM and Flash consumption are configurable for each NAS run, and fitness function weights are equivalent, i.e., $w_1 = w_2 = \cdots = w_5$ in all of our experiments.

Appendix C. Non-supernet configuration

The Non-supernet Data Aware NAS which we utilize in Section 5.5, is based on the state of the Data Aware NAS implementation at the *Removed for review period to preserve anonymity* commit hash in the repository *Removed for review period to preserve anonymity* along with a few modifications. The state of the repository at this commit hash can be found via the following URL: *Removed for review period to preserve anonymity*

The following modifications have been made to the state of the repository at this commit hash for the experiment described in Section 5.5:

- The steps per epoch parameter for training the NN model has been increased to 1000 steps.
- The number of training epochs has been increased to 25.
- Changed the NAS to run for 23 h rather than a fixed number of models.
- The data search space has been updated to that used in this paper.

The changes introduced to the training steps are done in an effort to align the training effort that goes into training both supernet- and non-supernet-based models. With these changes, a supernet-based model will be trained for 30,000 pre-training steps and 100 fine-tuning steps, whereas a non-supernet-based model will be trained for a total of 25,000 steps.

Data availability

The dataset used in this article is publicly available. Code can be found by following a link in the article.

References

[1] S.S. Saha, S.S. Sandha, M. Srivastava, Machine learning for microcontroller-class hardware: A review, IEEE Sens. J. 22 (22) (2022) 21362–21390.

[2] International Telecommunication Union, tinyML smart weather station challenge 2024: The next generation returns, 2024, URL https://aiforgood.itu.int/event/tinyml-smart-weather-station-challenge-2024-the-next-generation-returns/. (Accessed 10 October 2024).

[3] E. Njor, M.A. Hasanpour, J. Madsen, X. Fafoutis, A holistic review of the TinyML stack for predictive maintenance, IEEE Access (2024).

[4] V. Tsoukas, E. Boumpa, G. Giannakas, A. Kakarountas, A review of machine learning and tinyml in healthcare, in: Proceedings of the 25th Pan-Hellenic Conference on Informatics, 2021, pp. 69–73.

[5] NVIDIA Corporation, NVIDIA H100 tensor core GPU, 2025, URL https://www.nvidia.com/en-us/data-center/h100/. (Accessed 27 May 2025).

[6] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, P. Whatmough, Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers, Proc. Mach. Learn. Syst. 3 (2021) 517–532.

[7] M. Masiero, D. Schmidt, Apple iPhone 16 smartphone review – More innovations than one would think, 2024, URL https://www.notebookcheck.net/Apple-iPhone-16-smartphone-review-More-innovations-than-one-would-think.908118.0.html. (Accessed 27 May 2025).

[8] Arduino, Nano 33 BLE sense, 2024, URL https://docs.arduino.cc/hardware/nano-33-ble-sense.

[9] H. Benmeziane, K.E. Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, N. Wang, A comprehensive survey on hardware-aware neural architecture search, 2021, arXiv preprint arXiv:2101.09336.

[10] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han, et al., Mcunet: Tiny deep learning on iot devices, Adv. Neural Inf. Process. Syst. 33 (2020) 11711–11722.

[11] E. Njor, J. Madsen, X. Fafoutis, Data aware neural architecture search, 2023, arXiv preprint arXiv:2304.01821.

[12] Y. Koizumi, S. Saito, H. Uematsu, N. Harada, K. Imoto, ToyADMOS: A dataset of miniature-machine operating sounds for anomalous sound detection, in: 2019 IEEE Workshop on Applications of Signal Processing To Audio and Acoustics, WASPAA, IEEE, 2019, pp. 313–317.

[13] C. Banbury, E. Njor, M. Stewart, P. Warden, M. Kudlur, N. Jeffries, X. Fafoutis, V.J. Reddi, Wake vision: A large-scale, diverse dataset and benchmark suite for TinyML person detection, 2024, arXiv preprint arXiv:2405.00892.

[14] B. Zoph, Q. Le, Neural architecture search with reinforcement learning, in: International Conference on Learning Representations, 2017, URL https://openreview.net/forum?id=r1Ue8Hcxg.

[15] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8697–8710.

[16] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, F. Hutter, Nas-bench-101: Towards reproducible neural architecture search, in: International Conference on Machine Learning, PMLR, 2019, pp. 7105–7114.

[17] A. Brock, T. Lim, J. Ritchie, N. Weston, SMASH: One-shot model architecture search through HyperNetworks, in: International Conference on Learning Representations, 2018, URL https://openreview.net/forum?id=rydeCEhs-.

[18] E. Real, S. Moore, A. Selle, S. Saxena, Y.L. Suematsu, J. Tan, Q.V. Le, A. Kurakin, Large-scale evolution of image classifiers, in: International Conference on Machine Learning, PMLR, 2017, pp. 2902–2911.

[19] H. Liu, K. Simonyan, Y. Yang, DARTS: Differentiable architecture search, in: International Conference on Learning Representations, 2019, URL https://openreview.net/forum?id=S1eYHoC5FX.

[20] I. Fedorov, R.P. Adams, M. Mattina, P. Whatmough, Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers, Adv. Neural Inf. Process. Syst. 32 (2019).

[21] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[22] Y. Zhang, N. Suda, L. Lai, V. Chandra, Hello edge: Keyword spotting on microcontrollers, 2017, arXiv preprint arXiv:1711.07128.

[23] M. Gambella, A. Falcetta, M. Roveri, CNAS: Constrained neural architecture search, in: 2022 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2022, pp. 2918–2923.

[24] E. Liberis, Ł. Dudziak, N.D. Lane, μNas: Constrained neural architecture search for microcontrollers, in: Proceedings of the 1st Workshop on Machine Learning and Systems, 2021, pp. 70–79.

[25] L. Mocerino, A. Calimera, CoopNet: Cooperative convolutional neural network for low-power MCUs, in: 2019 26th IEEE International Conference on Electronics, Circuits and Systems, ICECS, IEEE, 2019, pp. 414–417.

[26] M. Mazumder, C. Banbury, X. Yao, B. Karlaš, W. Gaviria Rojas, S. Diamos, G. Diamos, L. He, A. Parrish, H.R. Kirk, et al., Dataperf: Benchmarks for data-centric ai development, Adv. Neural Inf. Process. Syst. 36 (2023) 5320–5347.

[27] A. Ratner, S.H. Bach, H. Ehrenberg, J. Fries, S. Wu, C. Ré, Snorkel: Rapid training data creation with weak supervision, in: Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases, Vol. 11, NIH Public Access, 2017, p. 269, (3).
[28] C. Northcutt, L. Jiang, I. Chuang, Confident learning: Estimating uncertainty in dataset labels, J. Artificial Intelligence Res. 70 (2021) 1373–1411.
[29] B. Settles, Active learning literature survey, 2009.
[30] C. Banbury, V. Janapa Reddi, P. Torelli, N. Jeffries, C. Kiraly, J. Holleman, P. Montino, D. Kanter, P. Warden, D. Pau, U. Thakker, a. torrini, j. cordaro, G. Di Guglielmo, J. Duarte, H. Tran, N. Tran, n. wenxu, x. xuesong, Mlperf tiny benchmark, in: J. Vanschoren, S. Yeung (Eds.), Proc. the Neural Inf. Process. Syst. Track Datasets Benchmarks 1 (2021) URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/da4fb5c6e93e74d3df8527599fa62642-Paper-round1.pdf.
[31] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen, et al., Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 12965–12974.
[32] A.M. Garavagno, D. Leonardis, A. Frisoli, ColabNAS: Obtaining lightweight task-specific convolutional neural networks following Occam's razor, Future Gener. Comput. Syst. 152 (2024) 152–159.
[33] H. Cai, L. Zhu, S. Han, Proxylessnas: Direct neural architecture search on target task and hardware, 2018, arXiv preprint arXiv:1812.00332.