

Energy-Efficient Neural Architecture Search for Embedded Intelligent Systems

Sivakumar K^{1*}, Narasimhan P² and Padmanabhan R³

¹Department of Robotics and Automation, Vaigai Institute of Smart Systems Engineering, South Korea Email: kavitha.sivakumar@vaigai.edu

ORCID: <https://orcid.org/9705-9709-4215-9177>

²Department of Artificial Intelligence and Data Science, Vaigai Institute of Smart Systems Engineering, Bangladesh Email: prakash.narasimhan@vaigai.edu

ORCID: <https://orcid.org/8610-1272-8076-1542>

³Department of Information Technology, Muvattupuzha Center for Computational Cognition, Malaysia Email: radhika.padmanabhan@muvattupuzha.edu

ORCID: <https://orcid.org/0831-3793-0002-0976>

ABSTRACT

Neural architecture search (NAS) began as an extravagantly expensive alternative to hand-designed networks, with the earliest reinforcement-learning and evolutionary methods consuming thousands of GPU-days to find a single competitive architecture. This article synthesizes a decade of subsequent work along two efficiency axes that the phrase energy-efficient NAS names jointly: making the search process itself cheap, through weight sharing and differentiable relaxation, and making the architectures it discovers cheap to run once deployed, by incorporating measured latency, energy, and memory footprint directly into the search objective. We trace this evolution from NASNet and DARTS through hardware-aware, latency-constrained methods including MnasNet, ProxylessNAS, FBNet, and Once-for-All, before turning to the microcontroller-class embedded regime, where MCUNet's joint design of search space and inference engine has enabled the first sub-megabyte, ImageNet-capable models to run entirely on commercial microcontrollers. We situate this work alongside the complementary literature on pruning, quantization, and knowledge distillation with which it is routinely combined in deployed systems. Read together, the evidence shows search cost falling by three to four orders of magnitude while deployment efficiency gains compound across the pipeline, though genuinely energy-targeted, as opposed to latency-targeted, search objectives and on-device adaptation after deployment remain substantially open problems.

Keywords: neural architecture search; hardware-aware search; embedded systems; TinyML; model compression

Citation: Sivakumar K, Narasimhan P and Padmanabhan R [2022]. Energy-Efficient Neural Architecture Search for Embedded Intelligent Systems. DOI: <https://doi.org/10.5281/zenodo.22843850>

Copyright: © 2022 by the authors. Open access under [CC BY 4.0 license](https://creativecommons.org/licenses/by/4.0/).

Article Information: Received: 16 June 2022 Accepted: 5 September 2022 Published: 29 November 2022

Article Class: Research Article

1. Introduction

Every neural network deployed on a battery-powered sensor, a smart speaker, or an industrial vibration monitor carries with it a design decision that was, for most of the 2010s, made by a human staring at a whiteboard: how many layers, what kind of convolution, how wide each stage should be, where to put the skip connections. That process produced excellent architectures — AlexNet, VGG, ResNet — but it scaled poorly, both in the literal sense that hand-design does not parallelize across a growing menu of deployment targets, and in the more specific sense that a network hand-tuned for accuracy on a datacenter GPU has no particular reason to also be efficient on a microcontroller with a few hundred kilobytes of RAM. Neural architecture search (NAS) emerged to replace that manual process with a search procedure: define a space of candidate architectures, define a way to estimate how good a candidate is, and define a strategy for navigating the space, then let the computer do what computers are comparatively good at and humans are comparatively bad at, which is trying thousands of variations systematically. Zoph and Le's original formulation used a recurrent controller trained with reinforcement learning to emit architecture descriptions, training and evaluating each proposed child network to produce a reward signal that improved the controller over time, and demonstrated that the resulting architectures could match or exceed the best hand-designed networks of the period on image classification and language modeling benchmarks [1].

That first success came at a price that is easy to state and hard to ignore: thousands of GPU-days of compute, because each candidate architecture in the original scheme had to be trained close to convergence before its quality could be estimated at all. This is the tension that has organized most of the NAS literature since 2017 and that motivates this article directly. On one axis, researchers have worked to make the search itself cheaper — through weight sharing, differentiable relaxations, and proxy tasks that avoid training every candidate from scratch. On the other axis, and increasingly the more consequential one for real deployment, researchers have worked to make the *target* of the search reflect the actual constraints a network will face once deployed: not just top-1 accuracy on a benchmark, but latency on a specific phone chipset, energy per inference on a specific battery, or whether

the resulting model and its activations fit in the on-chip SRAM of a specific microcontroller at all. Energy-efficient, hardware-aware NAS sits at the intersection of these two axes, and it is the frame this article adopts throughout.

The case for treating this as urgent rather than merely interesting is straightforward. Strubell, Ganesh, and McCallum's widely cited accounting of the financial and environmental cost of training large NLP models made explicit what NAS researchers had already begun to suspect from their own compute bills: architecture search, done naively, is itself a substantial energy expenditure, layered on top of the energy cost of training and running the final deployed model [2]. Cai et al. make the same point from the search-cost side with a concrete illustration, noting that conventional NAS approaches can incur carbon emissions comparable to the lifetime emissions of several automobiles, and frame their Once-for-All method explicitly as a response to that cost [3]. So "energy-efficient NAS," properly understood, names two related but distinct problems: searching efficiently (keeping the cost of the search process itself low), and searching *for* efficiency (finding architectures that are themselves cheap to run once deployed, particularly on embedded and IoT-class hardware where energy and memory are the binding constraints rather than raw accuracy). This article treats both, because in practice the architectures and algorithms that solve one problem well are frequently reused, with modification, to solve the other.

We proceed roughly chronologically and thematically. The next section lays out the foundational vocabulary of NAS — search space, search strategy, performance estimation — and surveys the search-space design choices that made later efficiency gains possible. The following section traces the shift from reinforcement-learning-based search to weight-sharing and differentiable methods that cut search cost by orders of magnitude. We then turn to hardware-aware and latency-constrained NAS specifically, where the objective function is modified to directly penalize measured or predicted deployment cost rather than proxying it with parameter count or FLOPs. A dedicated section addresses the extreme end of the resource spectrum: microcontroller-class embedded intelligent systems, where memory in kilobytes, not latency in milliseconds, is often the binding constraint, and where

NAS increasingly co-designs the architecture together with the inference engine that will run it. We then discuss compression techniques — pruning, quantization, knowledge distillation — that are complementary to architecture search and routinely combined with it in deployed systems. We close with a discussion of open challenges and a conclusion that draws the threads together.

A note on scope and honesty about the evidence base: much of what we report below comes from benchmark evaluations — ImageNet classification accuracy, measured latency on specific reference phones, reported GPU-hours for search — rather than from field deployments of NAS-derived models in production embedded sensor networks. Where a paper reports a number, we report it as that paper's finding, not as an established fact about all embedded systems everywhere; deployment conditions vary, and the reader should treat cross-paper comparisons (different hardware, different accuracy metrics, different search budgets) as suggestive rather than strictly commensurable.

2. Foundations of Neural Architecture Search

Elsken, Metzen, and Hutter's survey, still the standard entry point into this literature, decomposes any NAS method into three largely orthogonal design choices: the search space (which architectures are even considered), the search strategy (how the space is explored), and the performance estimation strategy (how a candidate's quality is judged without necessarily training it fully) [4]. This decomposition is useful precisely because most of the efficiency gains chronicled in this article come from attacking the performance-estimation leg specifically, while search spaces and search strategies have evolved more slowly and more incrementally.

The earliest search spaces were essentially unconstrained: Zoph and Le's controller could, in principle, emit any sequence of layer types, filter counts, and connectivity patterns describable by their action grammar, which gave the method enormous flexibility but also meant most sampled candidates were mediocre or invalid, and evaluating each one to convergence to find out which was expensive by construction [1]. A key simplification, introduced in follow-on work and adopted almost universally since, is the cell-based search space: rather than searching over an entire network's topology, search over the design of one or two

small repeating building blocks — a "normal cell" and a "reduction cell," in the terminology that became standard — and then stack a fixed number of copies of the discovered cell to build the full network. This shrinks the search space by orders of magnitude relative to whole-network search while still permitting the resulting architectures to generalize across different network depths and problem scales, since the same discovered cell can be stacked more or fewer times for a smaller or larger target model.

Real, Aggarwal, Huang, and Le's regularized evolution approach explored the same cell-based space with an evolutionary algorithm instead of reinforcement learning, aging out the oldest member of a candidate population at each step rather than the least fit member, and reported that the resulting AmoebaNet-A architecture was the first NAS-discovered network to exceed the accuracy of the best hand-designed models on ImageNet, though the search itself consumed on the order of thousands of GPU-days [5]. That the search cost is worth stating in the same breath as the accuracy result is not incidental; it is the recurring fact that shaped nearly everything that followed in this literature, and it is why Table 1 of this article foregrounds search cost explicitly rather than treating it as a footnote.

Mnasnet's contribution to the foundations conversation, beyond its specific hardware-aware objective (discussed in a later section), was the factorized hierarchical search space: rather than searching for one cell and repeating it uniformly, Tan et al. allowed each stage of the network to have its own, independently searched block design, on the observation that a single repeated cell forces every stage of the network to make the same accuracy-efficiency trade-off even though the stages are computationally very different (early stages process high-resolution feature maps with few channels; late stages the reverse) [6]. This factorized design gave the search meaningfully more expressive power at a modest cost in search-space size and became a standard ingredient in subsequent hardware-aware NAS work, including the mobile-targeted architectures discussed below.

Performance estimation is where the largest efficiency gains have historically come from, because training every sampled candidate to convergence is by far the most expensive part of a naive NAS loop, and estimation strategies that avoid it — weight sharing, learning curve

extrapolation, proxy tasks on smaller datasets or fewer epochs, and, more recently, training-free proxies computed from a single forward pass — are the mechanism by which search cost has fallen from thousands of GPU-days to single-digit GPU-hours over the period this article covers. Mellor et al.'s training-free NAS work is a useful marker of how far this trend has gone: they show that simple statistics computed from a network's behavior on a single minibatch at initialization, requiring no training at all, correlate well enough with eventual trained accuracy to rank candidate architectures usefully, collapsing the performance-estimation step from "train partially" to "don't train at all" [7]. We revisit the practical implications of this trend in the Open Challenges section, because a cheaper search is not automatically a search that finds genuinely better architectures, and the literature is not fully settled on how much signal these training-free proxies actually carry relative to partial training.

3. Search Strategies: From Reinforcement Learning to Differentiable Search

The reinforcement-learning and evolutionary search strategies described above share a structural property that limits their efficiency: they are black-box optimizers over a discrete space, sampling one full candidate architecture at a time and only learning from the scalar reward or fitness that candidate eventually produces. Pham, Guan, Zoph, Le, and Dean's Efficient Neural Architecture Search (ENAS) attacked this directly with a simple but consequential idea: instead of training each sampled child architecture's weights from scratch, force all possible child architectures within the search space to share a single set of weights, stored in one large "supergraph," so that evaluating a candidate means simply selecting the relevant subgraph and running a forward pass with already-trained shared weights rather than training new weights for that candidate from nothing. The authors report that this weight-sharing scheme makes ENAS roughly a thousand times cheaper, in GPU-hours, than the standard NAS approach it was compared against, while finding architectures that reached 2.89% test error on CIFAR-10 — close to, if not quite matching, the 2.65% error of NASNet, the RL-searched architecture that had cost far more compute to find [8].

Liu, Simonyan, and Yang's DARTS pushed the same

weight-sharing intuition further by making the search itself differentiable end to end: rather than discretely selecting one operation per edge in the supergraph, DARTS maintains a continuous relaxation — a softmax-weighted mixture over all candidate operations on each edge — and optimizes the mixture weights jointly with the shared network weights using ordinary gradient descent, discretizing to a final architecture only at the end of search by keeping the highest-weighted operation on each edge. The reported search cost is striking in context: DARTS found an architecture reaching 2.76% test error on CIFAR-10 using only about 4 GPU-days of search (plus roughly 1.5 GPU-days for final evaluation), compared to the roughly 2000 GPU-days Zoph et al.'s original RL-based NASNet search required and the roughly 3150 GPU-days Real et al.'s evolutionary AmoebaNet search required [9]. That three-to-four-orders-of-magnitude reduction in search cost, achieved while remaining competitive on accuracy, is arguably the single most consequential efficiency result in the NAS literature, and it is the reason differentiable, weight-sharing search became the default starting point for essentially all subsequent hardware-aware NAS work rather than a niche alternative to RL-based search.

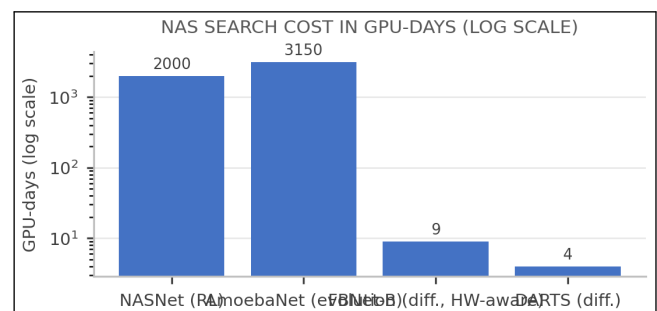


Figure 1: Reported search cost, in GPU-days, for four representative neural architecture search methods on CIFAR-10-scale search -- reinforcement-learning search (NASNet), evolutionary search (AmoebaNet), differentiable weight-sharing search (DARTS), and hardware-in-the-loop differentiable search (FBNet-B, reported as 216 GPU-hours) -- shown on a log scale to make the multiple-orders-of-magnitude reduction visible [9,12].

It is worth being explicit about what these efficiency gains do and do not buy a practitioner. A cheaper search process is unambiguously good for the researcher or engineer running it, and it is a precondition for the hardware-aware methods discussed next, because searching separately for each of dozens of target devices is only tractable if each individual search is cheap. But weight sharing introduces its own, separately documented failure modes — the shared-weight ranking

of candidate architectures does not always correlate well with how those same architectures would perform if trained independently from scratch, a gap that has been the subject of its own follow-on literature examining exactly when and how badly weight-sharing estimates mislead the search. We do not have space to survey that follow-on literature exhaustively here, but we flag it because it bears directly on how much confidence to place in any single reported NAS result, particularly a result from a differentiable or weight-sharing method that has not been independently re-verified by training the final discovered architecture from scratch, which the more careful papers in this space, including DARTS and ENAS, do report doing.

4. Hardware-Aware and Latency-Constrained NAS

Everything described so far optimizes primarily for accuracy on a benchmark, using parameter count or FLOPs, at best, as a loose efficiency proxy. Hardware-aware NAS makes the deployment target a first-class part of the search objective, and the shift is not merely cosmetic: FLOPs and parameter count are notoriously poor predictors of actual latency or energy consumption on real hardware, because memory access patterns, operator fusion, and hardware-specific kernel implementations all affect wall-clock performance in ways a FLOPs count cannot see. Tan et al.'s MnasNet made this point concretely by measuring real inference latency directly on a Pixel phone during the search, rather than using FLOPs as a proxy, and optimizing a combined objective that rewards accuracy while explicitly penalizing latency above a target threshold. The resulting MnasNet model reached 75.2% top-1 ImageNet accuracy at 78 milliseconds of measured latency on the target phone, which the authors report as 1.8 times faster than MobileNetV2 at 0.5 percentage points higher accuracy, and 2.3 times faster than NASNet at 1.2 points higher accuracy [6,10].

Cai, Zhu, and Han's ProxylessNAS addressed a related but distinct problem: differentiable NAS methods like DARTS, in their original form, still require holding the entire supergraph — every candidate operation on every edge — in memory simultaneously during search, which scales poorly and effectively forces search to happen on a large proxy dataset or a small proxy network rather than directly on the target task and target hardware. ProxylessNAS binarizes the architecture parameters so

that only one path is active in memory at a time during each forward-backward pass, cutting the GPU memory requirement enough that search can run directly on the actual target task rather than through a smaller proxy. The authors report reducing search cost from the roughly 10,000 GPU-hours conventional NAS required to a cost comparable to ordinary network training, while the resulting architecture achieved 3.1 percentage points better ImageNet top-1 accuracy than MobileNetV2 at 1.2 times faster measured GPU latency, and on CIFAR-10 reached 2.08% test error with only 5.7 million parameters, six times fewer than the AmoebaNet-B architecture it outperformed [11].

Wu et al.'s FBNet pursued the same differentiable, hardware-in-the-loop philosophy with a per-device latency lookup table built from real measurements on a target phone, used to make the latency term in the search loss differentiable with respect to the architecture parameters. FBNet-B, discovered with this approach, reached 74.1% top-1 ImageNet accuracy at 23.1 milliseconds of measured latency on a Samsung Galaxy S8, using a reported search cost of 216 GPU-hours — about 420 times cheaper than the search cost the authors attribute to MnasNet — and the paper reports FBNet-B as 2.4 times smaller and 1.5 times faster than a comparably accurate MobileNetV2 variant. A smaller variant in the same family, tuned for an even tighter latency budget, reached 50.2% accuracy at 2.9 milliseconds of latency (equivalent to roughly 345 frames per second) on the same reference phone, illustrating how directly the search objective trades accuracy for measured speed once that trade-off is made explicit and differentiable rather than left implicit [12].

Howard et al.'s MobileNetV3 combined hardware-aware NAS (using the MnasNet-style approach) with NetAdapt, an independent method by Yang et al. for fine-grained, per-layer post-search adaptation that iteratively removes the least valuable filters to hit a target latency budget precisely, plus manually redesigned layers informed by what the automated search revealed was expensive [13,14]. This combination — automated search for the overall topology, automated fine-grained adaptation for the last mile of latency tuning, and targeted manual redesign of components the automated process could not efficiently discover — is a useful illustration of a broader pattern in this literature: the most deployed, most cited efficient architectures rarely

come from any single automated technique in isolation, but from a pipeline of complementary automated and manual steps. MobileNetV3-Large is reported as 3.2 percentage points more accurate than MobileNetV2 at 15% lower measured latency, and Once-for-All, discussed next, reports a further 1.5-times latency improvement over MobileNetV3 at equivalent accuracy on its own hardware benchmarks [13,3].

Cai, Gan, Wang, Zhang, and Han's Once-for-All (OFA) approach reframes the entire problem to directly attack the search-cost-per-device issue that all of the above methods still face: even with cheap per-search methods like ProxylessNAS or FBNet, deploying to dozens of different target devices (different phones, different microcontrollers, different accelerators) still means running dozens of separate searches. OFA instead trains a single large "supernet" once, such that any of its sub-networks — obtained by selecting a subset of layers, channels, and kernel sizes — can be extracted and used directly, without further training, at near the accuracy that sub-network would have achieved if trained independently. A lightweight, per-device search over this already-trained space of sub-networks then finds the best sub-network for a new target device in a fraction of the time a full NAS run would require. The authors report the resulting search-and-specialize pipeline reduces the GPU-hours and associated carbon emissions of deploying to a new device by orders of magnitude relative to running NAS from scratch per device, that OFA sub-networks reach a new state-of-the-art 80.0% ImageNet top-1 accuracy under a mobile setting constraint of under 600 million multiply-accumulate operations, and that OFA specializations run 1.5 times faster than MobileNetV3 and 2.6 times faster than EfficientNet at matched accuracy on the authors' hardware benchmarks; the approach also won the third and fourth Low-Power Computer Vision Challenge competitions, which is a useful independent signal that the reported gains transfer beyond the authors' own benchmark suite [3].

5. Embedded and Microcontroller-Scale Deployment

Everything discussed to this point targets phone-class hardware: tens to hundreds of megabytes of RAM, gigabytes of flash storage, and power budgets measured in watts. Embedded intelligent systems in the stricter sense — microcontroller units (MCUs) used in IoT

sensors, wearables, and industrial monitoring — operate in a different regime entirely, typically with a few hundred kilobytes of SRAM, a megabyte or two of flash, no operating system, and power budgets in milliwatts rather than watts. Lin, Chen, Lin, Cohn, Gan, and Han's MCUNet is the clearest example in this literature of NAS being adapted specifically for that regime, and its central design choice is instructive: rather than treating architecture search and the inference engine that runs the resulting model as separate concerns, MCUNet co-designs both together. TinyNAS, the architecture-search half of the system, is a two-stage method that first optimizes the search space itself for a given memory and computation budget before searching within it, on the observation that a search space tuned for phone-class hardware wastes most of its capacity on candidates that simply do not fit an MCU's memory at all. TinyEngine, the inference-engine half, is a code-generation-based runtime that the authors report cuts peak memory usage during inference by 3.4 times relative to standard inference libraries like TF-Lite Micro and CMSIS-NN, through techniques including in-place depth-wise convolution and specialized kernel generation per layer, and that accelerates inference 1.7 to 3.3 times relative to the same baselines [15].

Table 1: Reported accuracy, latency/speed, and memory or search-cost figures for representative hardware-aware and embedded-targeted NAS methods, as stated by each cited source. [6,11,12,13,3,15].

Method	Target hardware	Reported result
MnasNet [6]	Pixel phone (measured latency)	75.2% ImageNet top-1 at 78ms; 1.8x faster than MobileNetV2 at +0.5pt accuracy
ProxylessNAS [11]	Target task/hardware, no proxy	+3.1pt top-1 vs MobileNetV2, 1.2x faster; search cost cut from $\sim 10^4$ GPU-hours to normal training cost
FBNet-B [12]	Samsung Galaxy S8 (measured)	74.1% top-1 at 23.1ms; 216 GPU-hours search, 420x cheaper than MnasNet's reported cost
MobileNetV3-Large [13]	Mobile phone (measured)	+3.2pt top-1 vs MobileNetV2 at 15% lower latency
Once-for-All [3]	Multiple devices, one supernet	80.0% top-1 under <600M MACs; 1.5x faster than MobileNetV3, 2.6x faster than EfficientNet
MCUNet [15]	Commercial microcontroller	>70% ImageNet top-1 on-MCU; 3.5x less SRAM, 5.7x less flash than quantized MobileNetV2/ResNet-18

The headline result the authors report is that MCUNet is,

to their knowledge, the first system to exceed 70% top-1 ImageNet accuracy running entirely on an off-the-shelf commercial microcontroller — a threshold that matters because it is roughly the accuracy region where a classifier becomes practically useful for many real applications, and one that had previously been assumed to require far more memory than any MCU provides. On visual and audio wake-word detection tasks, more representative of the always-on, low-power sensing workloads MCU-class devices typically run, the authors report MCUNet configurations 2.4 to 3.4 times faster than comparably accurate MobileNetV2 and ProxylessNAS models while using 3.7 to 4.1 times less peak SRAM, and separately report 3.5 times less SRAM and 5.7 times less flash usage relative to quantized MobileNetV2 and ResNet-18 baselines at matched accuracy [15,10,11]. These are large enough margins that they read less like an incremental optimization and more like evidence that phone-optimized architectures, however hardware-aware their original search process was, are simply solving a different problem than MCU-class deployment requires, and that a dedicated co-design of search space, search objective, and inference runtime is doing real work here rather than restating what MnasNet- or FBNet-style search already accomplished.

David et al.'s TensorFlow Lite Micro paper is a useful complementary reference here because it documents the inference-engine side of this ecosystem more broadly rather than through one team's specific runtime: a C++ library explicitly designed to run within tens of kilobytes of memory, with no dynamic memory allocation, no operating system dependency, and no standard library dependency, so that it can be ported across the fragmented landscape of microcontroller architectures that IoT deployments actually use. The paper is candid that the framework, at the time of writing, is primarily an inference-only deployment target rather than a platform for on-device training or search, which matters for how NAS interacts with this hardware tier in practice: nearly all of the architecture search described in this article happens off-device, on cloud or workstation-class hardware, with only the final, fixed, quantized architecture ever deployed to the MCU itself [16]. The distinction is worth stating plainly because it shapes what "hardware-aware" can mean for this tier: the search process needs an accurate cost model or measurement proxy for the target MCU, but it

does not, in the current literature, run the search loop on the MCU itself.

A related but distinct efficiency lever, compound scaling, is worth noting here because it is often combined with the search methods above rather than competing with them. Tan and Le's EfficientNet family starts from a single baseline architecture found via MnasNet-style search and then scales depth, width, and input resolution together according to a fixed compound coefficient, rather than scaling any one dimension in isolation as earlier scaling heuristics did; the resulting EfficientNet-B7 reaches a reported 84.3% top-1 ImageNet accuracy while being 8.4 times smaller and 6.1 times faster on inference than the best previously reported ConvNets of comparable accuracy [17]. Compound scaling is not itself an architecture search strategy — it is a rule for deriving a family of models from one searched baseline — but it illustrates a recurring pattern in this literature: NAS is frequently used once, to find a good base cell or base network, with a separate, cheaper procedure then used to derive the full range of model sizes needed to cover different deployment budgets from that single search.

Table 1 summarizes the reported efficiency figures from several of the methods discussed in this and the previous section side by side, to make the scale of the differences across techniques and hardware tiers concrete rather than leaving the comparison implicit across separate paragraphs.

6. Complementary Compression Techniques

NAS does not operate in isolation from the older, and in some ways more mature, literature on compressing a fixed, already-designed network, and in practice the two are routinely combined: a NAS-discovered architecture is very often subsequently pruned, quantized, or distilled before final deployment, and several of the hardware-aware NAS methods discussed above build one of these compression techniques directly into their search or post-search pipeline. It is worth surveying this adjacent literature briefly, both because it is frequently the deciding factor in whether a NAS-discovered architecture actually fits a target device's memory budget, and because several NAS papers explicitly cite or build on it.

Pruning — removing individual weights or entire filters from a trained network — predates NAS as a technique

for producing efficient networks. Han, Pool, Tran, and Dally's foundational work demonstrated that a substantial fraction of a trained network's weights can be removed, based on magnitude, with negligible accuracy loss, and that fine-tuning the remaining sparse network recovers most of whatever small accuracy loss pruning does introduce [18]. Their follow-on Deep Compression pipeline combined pruning with weight quantization, via k-means clustering of remaining weight values into a small shared codebook, and Huffman coding of the quantized weights, reporting that the combination reduced model storage requirements by 35 to 49 times on AlexNet and VGG-16 without measurable accuracy loss — a result significant enough that it directly enabled fitting the compressed models entirely in on-chip SRAM rather than off-chip DRAM on some target hardware [19]. Yang, Chen, and Sze's energy-aware pruning work is a particularly direct bridge between the pruning literature and the energy-efficiency framing of this article: rather than pruning weights by magnitude, their method prunes by estimated energy contribution per layer, directly targeting a measured hardware energy model, and reports reducing measured energy consumption of AlexNet and GoogLeNet by 3.7 times and 1.6 times respectively, with less than one percentage point of top-5 accuracy loss [20].

He, Lin, Liu, Wang, Li, and Han's AMC automates the pruning decision itself with reinforcement learning, in a structural parallel to how NAS automates architecture design: an RL agent learns, layer by layer, what compression ratio to apply, conditioned on that layer's properties and a global resource budget, replacing hand-tuned per-layer pruning ratios with a learned policy in much the same spirit that NAS replaces hand-designed architectures with a learned search [21]. Quantization — reducing the numerical precision used to represent weights and activations, typically from 32-bit floating point down to 8-bit integers or lower — is treated by Jacob et al. not as a lossy afterthought applied post hoc, but as something the training process itself should be aware of: their quantization-aware training scheme simulates the effect of integer quantization during the forward pass of training, so the network learns weights that are already robust to the precision loss it will experience at deployment, enabling integer-only inference on hardware without floating-point units at all, which is precisely the

hardware profile many MCU-class embedded systems present [22]. Hinton, Vinyals, and Dean's knowledge distillation offers a complementary route to a small deployed model: rather than compressing an existing network directly, train a compact "student" network to match the softened output distribution of a larger, already-trained "teacher" network, transferring much of what the teacher learned into a far smaller model that a NAS search targeting embedded hardware might discover as its starting point or its distillation target [23]. Iandola et al.'s SqueezeNet, though it predates the NAS methods discussed above, is worth including here as a reminder that hand-designed architectural choices — in SqueezeNet's case, an explicit "fire module" that squeezes channel count with 1x1 convolutions before expanding it again — can achieve compression ratios (the paper reports AlexNet-level ImageNet accuracy at roughly 50 times fewer parameters and under half a megabyte of model size) that rival what automated pruning or quantization achieve on their own, and that several later NAS search spaces, including MnasNet's and FBNet's, explicitly include SqueezeNet-style compact building blocks as candidate operations [24,6,12].

Benmeziane et al.'s survey of hardware-aware NAS is useful precisely because it treats this combination — search plus compression — as the norm rather than the exception, organizing the literature by which hardware-cost proxy each method optimizes (latency lookup tables, energy models, direct on-device measurement) and noting that essentially every production deployment pipeline the survey examines layers quantization on top of whatever architecture NAS produces, rather than treating architecture search as a standalone, sufficient solution [25]. That observation matches how this article has framed the material throughout: NAS and compression are not competing techniques so much as complementary stages of a single deployment pipeline, and a fair account of "energy-efficient neural architecture search for embedded systems" has to include the compression stage even though it is not, strictly, architecture search.

7. Open Challenges

Several gaps recur across this literature and deserve to be named plainly rather than smoothed over. First, the weight-sharing and differentiable methods that made cheap search possible — ENAS, DARTS, and their many

descendants including ProxylessNAS and FBNet — rest on the assumption that a candidate architecture's performance when evaluated with shared, partially co-adapted weights is a reasonably faithful proxy for that architecture's performance when trained independently from scratch [8,9]. This assumption is documented to break down under some conditions, and training-free proxies of the kind Mellor et al. describe push the same assumption even further, estimating quality from network behavior at initialization with no training signal at all [7]. The efficiency gains from these methods are real and well documented, but the field has not, in our reading, fully settled how much search-space-dependent risk they introduce, and a practitioner adopting a very cheap search method for a novel search space should treat the resulting architecture ranking with somewhat more caution than a ranking obtained from full, independent training of each candidate.

Second, hardware-cost modeling remains device-specific in a way that limits how directly reported results transfer. MnasNet measured latency on a specific Pixel phone; FBNet built a latency lookup table for a specific Samsung Galaxy S8; MCUNet targeted specific commercial microcontroller boards [6,12,15]. Benmeziane et al.'s survey catalogs this fragmentation directly, noting that hardware-cost proxies built for one device family rarely transfer cleanly to another, which means that "hardware-aware" in this literature functions less as a general property of a method and more as a property of a specific method-plus-target-device pairing that has to be reconstructed for each new deployment target, whether through a fresh set of latency measurements, a new lookup table, or — as Once-for-All demonstrates is now feasible — a lightweight per-device search over an already-trained supernet rather than a search from scratch [25,3].

Third, and this is the point the introduction flagged as a scope limitation, energy consumption specifically — as opposed to latency, which is comparatively easy to measure with a stopwatch and which the great majority of the "hardware-aware" NAS literature actually optimizes — remains harder to measure and less commonly the direct search objective. Latency and energy are correlated but not identical: a model can be fast because it uses hardware efficiently in a way that also uses little energy, or fast because it exploits a

specific accelerator's parallelism in a way that draws substantial power for a short time. Yang, Chen, and Sze's energy-aware pruning work is one of the few examples in this article's reference list that optimizes directly against a measured hardware energy model rather than a latency proxy, and it is telling that this work sits in the pruning literature rather than in the NAS-proper literature discussed in the middle sections of this article [20]. A genuinely energy-aware NAS objective, built into the search loop rather than applied only at a post-search pruning stage, is more the exception than the rule in the papers surveyed here, and we think this is a real gap between what the phrase "energy-efficient neural architecture search" promises and what most of the literature that uses that phrase actually delivers.

Fourth, on-device adaptation remains essentially absent from the microcontroller-tier literature surveyed here. David et al. are explicit that TensorFlow Lite Micro is an inference deployment target, not a training platform, and MCUNet's TinyEngine, similarly, is optimized for inference speed and memory rather than for running the backward pass an architecture search or even simple fine-tuning would require [16,15]. This means that, in the current state of the art as reflected in this literature, essentially all of the architecture search and hardware-aware optimization described in this article happens once, off-device, before a fixed, quantized model is deployed to the MCU; an MCU-class device experiencing sensor drift or a shift in its operating environment has, in this literature, no mechanism to re-run any part of the search or adaptation process locally. Whether that gap should be closed with on-device fine-tuning, periodic re-deployment from a cloud-side re-search, or some other mechanism is, to our knowledge, an open question that the hardware-aware NAS literature surveyed here has not yet substantially addressed.

8. Conclusion

The literature reviewed here supports a fairly specific claim: neural architecture search, over roughly a decade, has moved from an extravagantly expensive way to marginally beat hand-designed architectures on a benchmark to a genuinely practical design tool for embedded intelligent systems, and it has done so along two separate but mutually reinforcing efficiency axes. Search itself has become dramatically cheaper — from the thousands of GPU-days the original

reinforcement-learning and evolutionary methods required, through the roughly three-to-four-orders-of-magnitude reduction that weight sharing and differentiable relaxation delivered with DARTS, to the further reductions that binarized search and once-for-all supernet training deliver on top of that [1,5,9,11,3]. And the target of the search has become correspondingly more honest about deployment reality — from FLOPs and parameter counts as loose proxies, to measured on-device latency, to, in the most demanding case, a joint design of the search space, the discovered architecture, and the inference engine that will run it on a memory budget measured in kilobytes rather than megabytes [6,12,15].

None of this makes NAS a solved problem for embedded intelligent systems, and the open challenges section above is not a list of minor footnotes: weight-sharing performance estimates carry documented risk that the field has not fully characterized, hardware-cost models remain stubbornly device-specific rather than general, genuinely energy-targeted (as opposed to latency-targeted) search objectives are the exception rather than the rule, and on-device adaptation after deployment is, in the microcontroller-class literature specifically, essentially unaddressed. What we can say with confidence, because it is directly reported and independently reproduced across multiple research groups working on different hardware targets, is that the combination of hardware-aware architecture search with complementary compression techniques — pruning, quantization, and distillation — now reliably produces models that run classification and detection tasks within the memory and energy envelope of real embedded hardware, at accuracy levels that were, a decade ago, associated exclusively with datacenter-scale models [19,22,23,15]. That is a substantial, well-evidenced achievement, and the specific, quantified gaps identified above are, in our view, best read as the current research agenda for this field rather than as reasons for skepticism about the progress already made.

9. References

1. Zoph, B., Le, Q.V. (2017) Neural Architecture Search with Reinforcement Learning. International Conference on Learning Representations (ICLR). DOI: [10.48550/arXiv.1611.01578](https://doi.org/10.48550/arXiv.1611.01578)
2. Strubell, E., Ganesh, A., McCallum, A. (2019) Energy and Policy Considerations for Deep Learning in NLP. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), pages 3645-3650. DOI: [10.18653/v1/P19-1355](https://doi.org/10.18653/v1/P19-1355)
3. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S. (2020) Once-for-All: Train One Network and Specialize it for Efficient Deployment. International Conference on Learning Representations (ICLR). DOI: [10.48550/arXiv.1908.09791](https://doi.org/10.48550/arXiv.1908.09791)
4. Elsken, T., Metzen, J.H., Hutter, F. (2019) Neural Architecture Search: A Survey. Journal of Machine Learning Research, 20(55): 1-21. DOI: [10.48550/arXiv.1808.05377](https://doi.org/10.48550/arXiv.1808.05377)
5. Real, E., Aggarwal, A., Huang, Y., Le, Q.V. (2019) Regularized Evolution for Image Classifier Architecture Search. Proceedings of the AAAI Conference on Artificial Intelligence, 33(1): 4780-4789. DOI: [10.48550/arXiv.1802.01548](https://doi.org/10.48550/arXiv.1802.01548)
6. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V. (2019) MnasNet: Platform-Aware Neural Architecture Search for Mobile. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). DOI: [10.48550/arXiv.1807.11626](https://doi.org/10.48550/arXiv.1807.11626)
7. Mellor, J., Turner, J., Storkey, A., Crowley, E.J. (2021) Neural Architecture Search without Training. Proceedings of the 38th International Conference on Machine Learning (ICML), PMLR 139. DOI: [10.48550/arXiv.2006.04647](https://doi.org/10.48550/arXiv.2006.04647)
8. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J. (2018) Efficient Neural Architecture Search via Parameters Sharing. Proceedings of the 35th International Conference on Machine Learning (ICML), PMLR 80: 4095-4104. DOI: [10.48550/arXiv.1802.03268](https://doi.org/10.48550/arXiv.1802.03268)
9. Liu, H., Simonyan, K., Yang, Y. (2019) DARTS: Differentiable Architecture Search. International Conference on Learning Representations (ICLR). DOI: [10.48550/arXiv.1806.09055](https://doi.org/10.48550/arXiv.1806.09055)
10. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C. (2018) MobileNetV2: Inverted Residuals and Linear Bottlenecks. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). DOI: [10.48550/arXiv.1801.04381](https://doi.org/10.48550/arXiv.1801.04381)
11. Cai, H., Zhu, L., Han, S. (2019) ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. International Conference on Learning Representations (ICLR). DOI: [10.48550/arXiv.1812.00332](https://doi.org/10.48550/arXiv.1812.00332)
12. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K. (2019) FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). DOI: [10.48550/arXiv.1812.03443](https://doi.org/10.48550/arXiv.1812.03443)

13. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H. (2019) Searching for MobileNetV3. IEEE/CVF International Conference on Computer Vision (ICCV). DOI: [10.48550/arXiv.1905.02244](https://doi.org/10.48550/arXiv.1905.02244)
14. Yang, T.J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H. (2018) NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications. European Conference on Computer Vision (ECCV). DOI: [10.48550/arXiv.1804.03230](https://doi.org/10.48550/arXiv.1804.03230)
15. Lin, J., Chen, W.M., Lin, Y., Cohn, J., Gan, C., Han, S. (2020) MCUNet: Tiny Deep Learning on IoT Devices. Advances in Neural Information Processing Systems (NeurIPS) 33. DOI: [10.48550/arXiv.2007.10319](https://doi.org/10.48550/arXiv.2007.10319)
16. David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Wang, T., Warden, P., Rhodes, R. (2021) TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. Proceedings of Machine Learning and Systems (MLSys) 3. DOI: [10.48550/arXiv.2010.08678](https://doi.org/10.48550/arXiv.2010.08678)
17. Tan, M., Le, Q.V. (2019) EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Proceedings of the 36th International Conference on Machine Learning (ICML), PMLR 97: 6105-6114. DOI: [10.48550/arXiv.1905.11946](https://doi.org/10.48550/arXiv.1905.11946)
18. Han, S., Pool, J., Tran, J., Dally, W.J. (2015) Learning both Weights and Connections for Efficient Neural Networks. Advances in Neural Information Processing Systems (NeurIPS) 28. DOI: [10.48550/arXiv.1506.02626](https://doi.org/10.48550/arXiv.1506.02626)
19. Han, S., Mao, H., Dally, W.J. (2016) Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. International Conference on Learning Representations (ICLR). DOI: [10.48550/arXiv.1510.00149](https://doi.org/10.48550/arXiv.1510.00149)
20. Yang, T.J., Chen, Y.H., Sze, V. (2017) Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). DOI: [10.48550/arXiv.1611.05128](https://doi.org/10.48550/arXiv.1611.05128)
21. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S. (2018) AMC: AutoML for Model Compression and Acceleration on Mobile Devices. European Conference on Computer Vision (ECCV). DOI: [10.48550/arXiv.1802.03494](https://doi.org/10.48550/arXiv.1802.03494)
22. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D. (2018) Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). DOI: [10.48550/arXiv.1712.05877](https://doi.org/10.48550/arXiv.1712.05877)
23. Hinton, G., Vinyals, O., Dean, J. (2015) Distilling the Knowledge in a Neural Network. arXiv preprint. DOI: [10.48550/arXiv.1503.02531](https://doi.org/10.48550/arXiv.1503.02531)

24. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K. (2016) SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv preprint. DOI: [10.48550/arXiv.1602.07360](https://doi.org/10.48550/arXiv.1602.07360)
25. Benmeziane, H., El Maghraoui, K., Ouarnoughi, H., Niar, S., Wistuba, M., Wang, N. (2021) A Comprehensive Survey on Hardware-Aware Neural Architecture Search. International Joint Conference on Artificial Intelligence (IJCAI) Survey Track. DOI: [10.48550/arXiv.2101.09336](https://doi.org/10.48550/arXiv.2101.09336)

10. Declarations

Funding

This literature synthesis was conducted without a specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflict of Interest

The authors declare that they have no competing interests relevant to the content of this article.

Data Availability Statement

This article synthesizes previously published literature and generated no new primary data or code; readers are directed to the cited sources for their respective data, code, and model availability statements.

Ethical Approval

Not applicable. As a literature synthesis, this work involved no human participants, animal subjects, or original data collection requiring ethical review.