

# 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 CONSTRAINT-GUIDED HARDWARE-AWARE NAS THROUGH GRADIENT MODIFICATION

Anonymous authors

Paper under double-blind review

## ABSTRACT

Neural Architecture Search (NAS), particularly gradient-based techniques, has proven highly effective in automating the design of neural networks. Recent work has extended NAS to hardware-aware settings, aiming to discover architectures that are both accurate and computationally efficient. Many existing methods integrate hardware metrics into the optimization objective as regularization terms, which introduces differentiability requirements and hyperparameter tuning challenges. This can either result in overly penalizing resource-intensive architectures or architectures failing to meet the hardware constraints of the target device. To address these challenges, we propose CONNAS, a novel gradient-based NAS framework that enforces hardware constraints directly through gradient modification. This approach eliminates the need for differentiable hardware metrics and regularization weights. The novelty in CONNAS lies in modifying gradients with respect to architectural choices, steering the search away from infeasible architectures while ensuring constraint satisfaction. Evaluations on the NATS-Bench benchmark demonstrate that CONNAS consistently discovers architectures that meet the imposed hardware constraints while achieving performance within just 0.14% of the optimal feasible architecture. Additionally, in a practical deployment scenario, CONNAS outperforms handcrafted architectures by up to 1.55% in accuracy under tight hardware budgets.

## 1 INTRODUCTION

Deep neural networks have proven to be very successful in numerous applications ranging from image recognition (Krizhevsky et al., 2017) and speech recognition (Hinton et al., 2012) to time series segmentation (Lea et al., 2017). More recently, the use of deep neural networks on constrained hardware has gained significant interest in the context of machine learning on the edge (EdgeML), where resource-constrained devices such as mobile devices, embedded computers, and microcontrollers are used to perform inference tasks. These devices often have limited computational power, memory, and energy resources. As a result, it is crucial to design efficient neural network architectures that can operate effectively within these hardware constraints.

A key challenge in EdgeML is designing neural network architectures that meet hardware constraints while maintaining high performance. In recent years, Neural Architecture Search (NAS) was introduced to automate the design of neural networks, and to potentially find novel architectures that outperform manually designed models by experts (Elsken et al., 2019b). In NAS, a search algorithm is used to explore a search space of possible neural network architectures, aiming to find the best-performing architecture for a given task. Models originating from NAS techniques have already outperformed human-designed models on tasks such as image classification (Real et al., 2017; Zoph et al., 2018) and semantic segmentation (Chen et al., 2018). Based on the type of search algorithm used, NAS methods can be categorized into four groups: *(i)* Reinforcement Learning (RL) (Zoph & Le, 2017; Zoph et al., 2018; Baker et al., 2017), *(ii)* Evolutionary Algorithms (EA) (Real et al., 2017; Salimans et al., 2017), *(iii)* Bayesian Optimization (White et al., 2021), *(iv)* and Gradient-based methods (Liu et al., 2019). The latter has gained significant popularity in recent years due to its efficiency and ability to scale to large search spaces compared to the other methods.

Recently, the use of NAS for designing more efficient neural network architectures has been increasingly explored. Prior work, specifically on gradient-based NAS, often integrates hardware

metrics as regularization terms into the optimization function (Cai et al., 2019; Wu et al., 2019; Wan et al., 2020). While this can help guide the search toward hardware-friendly architectures, these approaches often face several challenges. First, to be compatible with gradient-based optimization, the hardware metrics must be differentiable. Second, each regularization term requires appropriate weighting, introducing additional hyperparameters that demand careful tuning. Improper weighting can either overly penalize resource-intensive architectures (resulting in simpler models with suboptimal performance) or fail to enforce the hardware constraints (yielding architectures unsuitable for deployment on the target device). Consequently, these methods often require multiple search runs with varying weights assigned to the hardware-related terms until a feasible architecture is found, a process that can be both tedious and time-consuming.

To overcome these limitations, we introduce CONNas, a novel hardware-aware, gradient-based NAS algorithm. Our work makes the following key contributions:

- **Direct constraint enforcement:** Unlike prior approaches that incorporate hardware metrics as regularization terms in the loss function, CONNas modifies gradients with respect to architectural choices to directly enforce hardware constraints, effectively steering the search away from infeasible architectures.
- **No need for differentiable:** Our method eliminates the requirement for differentiable hardware metrics and the associated techniques needed to enforce differentiability.
- **Explicit constraint specification:** CONNas allows for the explicit definition of hardware constraints without relying on hyperparameter tuning to balance the importance of hardware metrics.

We evaluate CONNas on the NATS-Bench benchmark (Dong et al., 2021), demonstrating its ability to systematically find architectures that satisfy various hardware constraints while achieving performance close to the optimal architecture available within the search space, reaching as little as a 0.14% difference in accuracy. Our experiments show that CONNas significantly outperforms existing baseline methods, which often struggle to find feasible architectures under strict hardware constraints. Additionally, we validate CONNas in a practical use case, where it successfully discovers high-performing architectures under strict hardware constraints (no more than a model size of 64kB and 18kB memory usage), consistently outperforming the best handcrafted architectures by up to 1.55% accuracy.

## 2 RELATED WORK

**Hardware-aware NAS.** Over the years, the focus of NAS has shifted from discovering top-performing novel architectures to emphasizing computational efficiency. The design of neural networks has increasingly been guided by hardware-aware considerations, particularly in the context of deployment on resource-constrained devices. Early works primarily focused on optimizing hardware metrics which could be estimated based on the type of operations found in the model, such as the number of Floating Point Operations (FLOPs) (Xie et al., 2019; Zhou et al., 2018). However, other works have proposed to rely on real hardware metrics for more representative evaluation, which are generally obtained through look-up tables or regressors trained on real on-device benchmarks of various neural operations (Cai et al., 2019; Wu et al., 2019; Wan et al., 2020; Tan et al., 2019; Hu et al., 2020). Additionally, other contributions have a more specific focus on NAS techniques and search spaces tailored for ultra-constrained devices, such as microcontrollers (Lin et al., 2020; Liberis et al., 2021).

The most straightforward way to regularize hardware-related objectives is by incorporating hardware metrics as additional terms in the optimization function (Xie et al., 2019; Cai et al., 2019; Wu et al., 2019; Dong & Yang, 2019a; Wan et al., 2020; Bender et al., 2020; Hu et al., 2020). A weighting factor is then introduced to adjust the relative importance of these hardware metrics in comparison to the task-specific loss, effectively determining whether more constrained architectures should be favored. However, this fixed approach does not offer a direct mechanism for enforcing hardware constraints. Instead, it merely guides the search toward architectures with more favorable hardware metric values (typically simpler architectures with limited representational capacity) regardless of whether the current architecture actually satisfies the imposed constraints. In response, Tan et al. (2019); Zhou et al. (2018); Dong & Yang (2019a); Bender et al. (2020) propose dynamic optimization functions where hardware-related terms are adjusted depending on whether the current

108 architecture meets the hardware constraints. The rationale behind this approach is to find architectures  
 109 that are close to the constraint boundary, which makes it possible to explore more complex  
 110 architectures that still satisfy the constraints. This is in contrast to methods with a fixed loss term,  
 111 which may limit the search to simpler architectures. Meanwhile, other techniques (Elsken et al.,  
 112 2019a; Liberis et al., 2021), typically non-gradient-based methods, use a selection procedure based  
 113 on hardware metric values, selecting only architectures that satisfy hardware constraints for further  
 114 optimization.

115 **Gradient-based NAS.** Gradient-based NAS techniques, originally proposed by DARTS (Liu et al.,  
 116 2019), have been used by recent works to efficiently search for neural network architectures. In  
 117 gradient-based NAS, all possible architectures are represented within an over-parameterized net-  
 118 work, where each candidate path is associated with a continuous architecture weight  $\alpha$ . This formu-  
 119 lation leads to a bi-level optimization problem, in which both the architecture  $\alpha$  and its weights  $w$   
 120 are jointly optimized using gradient descent. Traditionally, the loss function would be defined as the  
 121 task-specific loss, such as cross-entropy for classification tasks. However, in hardware-aware NAS,  
 122 the loss function is augmented with regularization terms to account for hardware metrics.

123 While hardware metrics for individual architectures are not differentiable, state-of-the-art techniques  
 124 ensure that these metrics can be fully factorized over the architecture parameters. This enables ex-  
 125 pressing hardware metrics as a weighted sum of contributions from each candidate operation in  
 126 the search space, thereby making them differentiable with respect to the architecture parameters.  
 127 However, this approach requires knowledge of each operation’s contribution to the overall hardware  
 128 metric, which becomes computationally intractable in large search spaces. To address this, Xie et al.  
 129 (2019) propose approximating the hardware-related terms using Monte Carlo estimation. Neverthe-  
 130 less, if the sampling distribution is poorly calibrated, the resulting estimates may be inaccurate.

131 **Constraint-aware Training.** Some recent NAS approaches incorporate hardware constraints di-  
 132 rectly into the optimization process, though this area remains relatively unexplored. HardCore-  
 133 NAS (Nayman et al., 2021) addresses hard-constrained optimization by adapting the Frank-Wolfe  
 134 algorithm to search for sub-networks within a pre-trained one-shot model. HDX (Hong et al., 2022)  
 135 introduces hardware constraints by adding hardware metrics as regularization terms and modifying  
 136 gradients based on differentiated hardware metrics. Similarly, Constraint Guided Gradient Descent  
 137 (CGGD) (Van Baelen & Karsmakers, 2023), although not a NAS algorithm, enforces inequality  
 138 constraints during training through gradient modification without penalty terms, proposing an up-  
 139 date scheme that guides optimization toward weights satisfying the specified constraints. Building  
 140 on these ideas, our work explores how such gradient-based enforcement can directly impose hard-  
 141 ware constraints during the NAS search process, without requiring additional regularization terms  
 142 or differentiable hardware metrics.

### 3 METHOD

#### 3.1 PROBLEM FORMULATION

148 The search space  $S$  is modeled as an over-parameterized neural network that encompasses all pos-  
 149 sible architectures. This is achieved by defining, for each layer in the network, a set of candidate  
 150 operations (e.g., convolutions with varying kernel sizes) that can be used to construct a neural net-  
 151 work. Each candidate operation is associated with a continuous architecture weight  $\alpha$ , which enables  
 152 the use of gradient descent to jointly optimize both operation weights  $w$  and the architecture weights  
 153  $\alpha$ . A final architecture  $A$  can then be derived by selecting a subset of candidate operations based on  
 154 the learned architecture weights  $\alpha$ . More formally, let the over-parameterized network  $S$  be defined  
 155 as a directed acyclic graph (DAG) with  $L$  edges where each edge  $e_l$  is associated with a set of candi-  
 156 date operations  $\mathcal{O}_l = \{o_{l,1}, \dots, o_{l,N}\}$  with corresponding architecture weights  $\alpha_l = [\alpha_{l,1}, \dots, \alpha_{l,N}]$   
 157 and operation weights  $w_l = [w_{l,1}, \dots, w_{l,N}]$ . An architecture  $A \in S$  is derived by selecting, for  
 158 each edge  $e_l$ , the operation with the highest architecture weight, i.e.,  $A = (o_{1,k_1}, \dots, o_{L,k_L})$  where  
 159  $k_l = \arg \max_k \alpha_{l,k}$ . Let  $c_k$  denote the  $k$ -th hardware metric function (e.g., number of parameters),  
 160 and  $b_k$  its upper bound. The set of hardware constraints is defined as  $\mathcal{C} = \{(c_1, b_1), \dots, (c_M, b_M)\}$ ,  
 161 where each constraint is expressed as  $c_k(A) \leq b_k$ . The search process can then be formulated as  
 follows:

$$\arg \min_{w, \alpha} \mathcal{L}_{\text{task}}(w, \alpha) \quad \text{s.t.} \quad c_k(A) \leq b_k, \quad k = 1, \dots, M \quad (1)$$

In other words, the goal is to find architecture weights  $\alpha$  that yield an architecture  $A(\alpha)$  minimizing the task-specific loss  $\mathcal{L}_{\text{task}}$ , while ensuring that the resulting architecture lies within the feasibility region  $FR$ , defined as the subset of architectures that satisfy all hardware constraints in  $\mathcal{C}$ . Formally, we aim to find  $A(\alpha) \in FR$ , or at least ensure that the sampled architectures  $A^k$  generated during optimization converge to  $FR$ , i.e.,  $\lim_{k \rightarrow \infty} d(A^k, FR) = 0$ .

The value for a hardware metric is obtained using  $c_k(A)$ , where  $c_k$  may be implemented as a look-up table or a learned regressor. Unlike prior work on hardware-aware, gradient-based NAS (Cai et al., 2019; Wu et al., 2019; Wan et al., 2020; Dong & Yang, 2019a), we do not impose that the hardware metrics are made differentiable with respect to the architecture weights  $\alpha$ .

### 3.2 GRADIENT UPDATE SCHEME

We adopt the gradient update scheme as proposed in CGGD (Van Baelen & Karsmakers, 2023) to enforce the hardware constraints  $c_k(A) \leq b_k$ ,  $k = 1, \dots, M$ . However, unlike CGGD, which aims to enforce constraints on the model's output predictions, we use this approach to manipulate the architecture weights  $\alpha$  to steer the search process towards feasible architectures  $A$ . Specifically, each architecture weight  $\alpha_{l,j}$  is updated as follows:

$$\alpha_{l,j} \leftarrow \alpha_{l,j} - \eta_\alpha \cdot (\nabla_{\alpha_{l,j}} \mathcal{L}_{\text{task}}(w, \alpha) + R \cdot \mathbf{dir}_{\mathcal{C}}(\alpha_l) \cdot \max \{ \|\nabla_{\alpha_{l,j}} \mathcal{L}_{\text{task}}(w, \alpha)\|, \epsilon \}) \quad (2)$$

Here,  $R$  is a rescale factor that determines the strength of the constraints enforcement, and  $\mathbf{dir}_{\mathcal{C}}(\alpha_l) = [\mathbf{dir}_{\mathcal{C}}(\alpha_{l,1}), \dots, \mathbf{dir}_{\mathcal{C}}(\alpha_{l,N})]$  is a unit vector indicating the direction in which the architecture weights should be modified to satisfy the constraints. An arbitrarily small constant  $\epsilon$  is added to allow adjustments even when the loss gradient is zero.

### 3.3 GRADIENT DIRECTION

The choice of  $R$  and  $\mathbf{dir}_{\mathcal{C}}(\alpha_l)$  is crucial, as it determines whether the architecture weights are modified in a way that effectively reduces the hardware constraint violations, and ultimately leads to convergence to a feasible architecture. Van Baelen & Karsmakers (2023) proved that convergence towards the  $FR$  is guaranteed if  $R > 1$  and  $\mathbf{dir}_{\mathcal{C}}(\alpha_l)$  is chosen to be the shortest path with respect to the Euclidean distance to the feasibility region. They also showed that performance is not highly sensitive to the exact value of  $R$ , as long as it is strictly larger than 1. In other words, this eliminates the need for additional hyperparameter tuning, unlike prior work that relies on weighted regularization terms.

However, computing the exact shortest path is often intractable because it requires knowing the hardware metrics for all possible architectures in the search space, or at least that these metrics are fully factorizable across all edges. To address this, we propose a heuristic approach to approximate the direction of the global shortest path by calculating the shortest path for each edge  $e_l$  independently. However, this means that the same convergence guarantees as in Van Baelen & Karsmakers (2023) do not strictly hold. Nevertheless, our experiments (Section 4.1) demonstrate that this heuristic is effective in practice.

For each edge  $e_l$ , we first evaluate the hardware metric values of all candidate operations  $\mathcal{O}_l$  under the current architecture configuration. Then, we construct a set of candidate architectures  $\mathcal{A}_l = \{A_{l,1}, \dots, A_{l,N}\}$  by replacing the  $l$ -th operation with  $o_{l,j}$  while keeping the rest fixed:

$$A_{l,j} = (o_{1,k_1}, \dots, o_{l,j}, \dots, o_{L,k_L}) \quad (3)$$

We then evaluate  $c_k(A_{l,j})$  for each candidate and constraint  $(c_k, b_k) \in \mathcal{C}$  in order to determine  $\mathbf{dir}_{\mathcal{C}}^k(\alpha_l)$ , which is the direction to the  $FR$  for constraint  $c_k(A_{l,j}) \leq b_k$ . Based on whether the candidate architectures  $\mathcal{A}_l$  satisfy the hardware constraint, we distinguish three cases:

**1. All candidates on  $e_l$  satisfy the hardware constraint** ( $\forall j \in \{1, \dots, N\}$ ,  $c_k(A_{l,j}) \leq b_k$ ). In this case, we set  $\mathbf{dir}_{\mathcal{C}}^k(\alpha_l) = 0$  for all candidates, as no modification is needed to satisfy the constraint for this edge.

**2. Some candidates on  $e_l$  violate the hardware constraint** ( $\exists j$ ,  $c_k(A_{l,j}) > b_k$ ). In this situation, the direction is computed by aggregating the unit vectors that point away from the candidates that violate the constraint, while pointing towards those that satisfy it. More specifically, we first identify the set of candidates that satisfy the constraint:

$$\mathcal{F} = \{j \mid c_k(A_{l,j}) \leq b_k\} \quad (4)$$

216 For each pair  $(j, m)$  where  $j \in \mathcal{F}$  and  $m \notin \mathcal{F}$ , we compute the unit vector that points from candidate  
 217  $m$  to candidate  $j$ , as shown in Figure 1a:

$$219 \quad \mathbf{u}_{j,m} = [\underbrace{0, \dots, 0}_{j-1}, \underbrace{1, 0, \dots, 0}_{m-j-1}, \underbrace{-1, 0, \dots, 0}_{N-m}] / \sqrt{2} \quad (5)$$

$$220$$

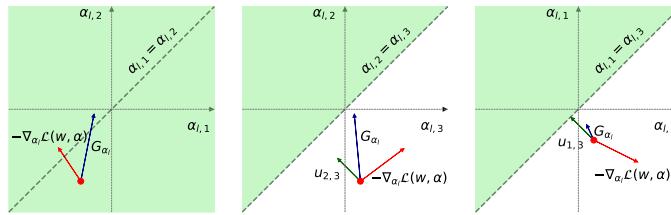
221 Then, the final direction  $dir_c^k(\alpha_l)$  is determined by summing these unit vectors and normalizing it  
 222 back to a unit vector:

$$223 \quad dir_c^k(\alpha_l) = \sum_{j \in \mathcal{F}} \sum_{m \notin \mathcal{F}} \mathbf{u}_{j,m} / \left\| \sum_{j \in \mathcal{F}} \sum_{m \notin \mathcal{F}} \mathbf{u}_{j,m} \right\| \quad (6)$$

$$224$$

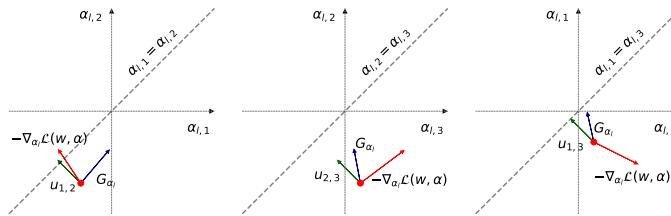
225 **3. No candidate on  $e_l$  satisfies the hardware constraint** ( $\forall j \in \{1, \dots, N\}$ ,  $c_k(A_{l,j}) > b_k$ ). In  
 226 this case, the direction is computed in such that candidates with higher hardware metric values are  
 227 penalized while those with lower values are rewarded. First, the candidates are ranked based on their  
 228 hardware metric values. Then, for each ranked candidate, we compute unit vectors  $\mathbf{u}_{j,m}$  (as defined  
 229 in Equation 5) that point from candidates with higher hardware metric values to those with lower  
 230 ones. Specifically:

- 232 • First, we compute the unit vectors between the candidate with the highest hardware metric value  
 233 and all other candidates.
- 234 • Then, we compute the unit vectors between the two candidates with the highest hardware metric  
 235 values and all others.
- 236 • This process is repeated until all candidates, except the one with the lowest hardware metric value,  
 237 have been considered, as shown in Figure 1b. Finally, the unit vectors are aggregated in the same  
 238 manner as in Equation 6.



240  
 241  
 242  
 243  
 244  
 245  
 246  
 247  
 248  
 249  
 250  
 251  
 252  
 253  
 254  
 255  
 256  
 257  
 258  
 259  
 260  
 261  
 262  
 263  
 264  
 265  
 266  
 267  
 268  
 269

(a) Both  $A_{l,1}$  and  $A_{l,2}$  satisfy the constraint,  $A_{l,3}$  violates it, so the resulting gradient  $G_{\alpha_l}$  is decreased for  $\alpha_{l,3}$  and increased for  $\alpha_{l,1}$  and  $\alpha_{l,2}$ .



(b) No candidate for  $e_j$  satisfies the constraint, so the resulting gradient  $G_{\alpha_l}$  is increased / decreased based on the ranked hardware metric values of each candidate.

261 Figure 1: Architecture space for a single edge  $e_l$  with three candidate operations, parameterized by  
 262  $\alpha_{l,1}, \alpha_{l,2}, \alpha_{l,3}$ , where  $c_k(A_{l,1}) < c_k(A_{l,2}) < c_k(A_{l,3})$ . Each subfigure shows a projection onto the  
 263 plane defined by two architecture weights. The shaded area represents the feasibility region  $FR$  for  
 264  $c_k$ .  $G_{\alpha_l}$  is the gradient after modification projected into each plane.

### 3.4 CONSTRAINT-GUIDED HARDWARE-AWARE NAS

268 We refer to Algorithm 1 for the modified training procedure. Within each batch, the optimization of  
 269 operation weights  $w$  and architecture weights  $\alpha$  is performed in an interleaved manner, as proposed  
 in prior work (Cai et al., 2019; Wu et al., 2019; Wan et al., 2020; Dong & Yang, 2019a;b). After

---

270           **Algorithm 1** Training procedure under hardware constraints

---

271           Input: Search space  $S$ , current architecture  $A$  derived from  $\alpha$ , hardware constraints  $\mathcal{C}$ , rescale factor  $R$ ,  
 272           epsilon  $\epsilon$ , learning rates  $\eta_w, \eta_\alpha$ , loss function  $\mathcal{L}_{\text{task}}$ , batch of data  $(X, y)$   
 273            $w \leftarrow w - \eta_w \cdot \nabla_w \mathcal{L}_{\text{task}}(w, \alpha)$  {Update operation weights}  
 274            $\mathbf{G}_\alpha = \nabla_\alpha \mathcal{L}_{\text{task}}(w, \alpha)$  {Store gradients w.r.t. architecture weights}  
 275            $\text{dirs}_c = []$  {For each unsatisfied hardware constraint, compute direction to feasibility region}  
 276           **for**  $(c_k, b_k) \in \mathcal{C}$  **do**  
 277            **if**  $c_k(A) > b_k$  **then**  
 278            Append  $\text{calculate\_gradient\_direction}(A, c_k)$  to  $\text{dirs}_c$   
 279            **end if**  
 280           **end for**  
 281            $\text{dir}_c(\alpha) = \sum_{\text{dir}_c^k \in \text{dirs}_c} \text{dir}_c^k / \left\| \sum_{\text{dir}_c^k \in \text{dirs}_c} \text{dir}_c^k \right\|$  {Aggregate directions into single unit vector}  
 282            $\mathbf{G}_\alpha \leftarrow \mathbf{G}_\alpha + R \cdot \text{dir}_c(\alpha) \cdot \max \{ \|\mathbf{G}_\alpha\|, \epsilon \}$  {Modify gradient toward the feasibility region}  
 283            $\alpha \leftarrow \alpha - \eta_\alpha \mathbf{G}_\alpha$  {Update architecture weights}

---

284  
 285           optimizing  $w$ , the gradient  $\nabla_\alpha \mathcal{L}_{\text{task}}(w, \alpha)$  is computed, but instead of immediately updating  $\alpha$ , it  
 286           is stored for later modification. Then, for each hardware metric  $c_k$  and its associated constraint  
 287            $c_k(A) \leq b_k$ , the current architecture  $A$  is checked for violations. If a constraint is violated, the  
 288           direction for modification  $\text{dir}_c^k(\alpha)$  is computed as described in Section 3.3. All directions are  
 289           summed and normalized to a unit vector to obtain  $\text{dir}_c(\alpha)$ . Finally, the stored gradient is modified,  
 290           as explained in Section 3.2, and used to update the architecture weights  $\alpha$ .

291  
 292           **4 EXPERIMENTS AND RESULTS**

293  
 294           We evaluate the effectiveness of CONAS on two benchmark tasks: the NATS-Bench topology and  
 295           size search spaces (Section 4.1). In addition, we apply our method to a practical use case involving  
 296           edge-based condition monitoring of induction motors, where the goal is to discover architectures  
 297           suitable for deployment across diverse hardware configurations (Section 4.2).

298  
 299           **4.1 EXPERIMENTS ON NATS-BENCH**

300           NATS-Bench, proposed by Dong et al. (2021), is a unified benchmark for NAS designed for image  
 301           classification tasks, including two search spaces: the topology search space and the size search space.  
 302           The topology search space is a cell-based search space, which contains 15,625 unique architectures,  
 303           while size search space is a layer-wise search space, consisting of 32,768 unique architectures. We  
 304           refer to Dong et al. (2021) for a detailed description of the search spaces. A key advantage of NATS-  
 305           Bench over other NAS benchmarks is that it provides test accuracies for all architectures across  
 306           both search spaces on CIFAR-10, CIFAR-100, and ImageNet16-120. This enables rapid evaluation  
 307           without the need for retraining. Additionally, we combine both real on-device measurements and  
 308           proxy-based hardware metrics to define hardware constraints for our experiments. Specifically, we  
 309           use on-device measurements of inference latency and energy usage on an NVIDIA Jetson TX2  
 310           Edge GPU (NVIDIA Inc.) for the topology search space, as provided by HW-NAS-Bench (Li et al.,  
 311           2021). For the size search space, we incorporate widely used proxy-based metrics, including the  
 312           number of parameters, FLOPs, and peak memory usage, all of which can be computed directly from  
 313           the architecture definition. A detailed description of how these proxy-based hardware metrics are  
 314           calculated is provided in Appendix A. Importantly, our approach is flexible and can integrate any  
 315           other hardware metric, regardless of how it is obtained or computed.

316           We use a Gumbel-Softmax (Jang et al., 2017), as proposed in Dong & Yang (2019a); Wu et al.  
 317           (2019); Wan et al. (2020), to relax the categorical distribution of the candidate operations  $\mathcal{O}_l$ . At  
 318           each edge  $e_l$ , the output  $y_l$  is computed as a weighted sum of the outputs produced by all candidate  
 319           operations:

$$y_l = \sum_j^N \frac{\exp((\log(\alpha_{l,j}) + g_{l,j})/\tau)}{\sum_j^N \exp((\log(\alpha_{l,j}) + g_{l,j})/\tau)} \cdot o_{l,j}(\mathbf{x}_l) \quad (7)$$

320           where  $g_{l,j} \sim \text{Gumbel}(0, 1)$  and temperature  $\tau$  controls the smoothness of the distribution. A higher  
 321           temperature is used at the beginning of the search to encourage exploration, making the distribu-  
 322           tion closer to uniform. As the search progresses, the temperature is gradually lowered, making the

324  
 325 **Table 1: Overview of baseline methods used for comparison.** Here,  $c'_k(\alpha)$  is the min-max normalization  
 326 of  $c_k(\alpha)$ . The term  $b_k$  represents the upper bound associated with  $c_k$ .  $\lambda_{\text{hardware}}$  is a weighting factor. We  
 327 use  $\lambda_{\text{hardware}} = 1$  for ProxylessNAS, TAS, TF-NAS and HDX, and  $\lambda_{\text{hardware}} = 0.6$  for FBNet. Additional  
 328 experiments with varying weighting factors for  $\lambda_{\text{hardware}}$  are provided in Appendix D.

Name	Regularization
ProxylessNAS (Cai et al., 2019)	$\mathcal{L}(w, \alpha) = \mathcal{L}_{\text{task}}(w, \alpha) + (\lambda_{\text{hardware}}/M) \cdot \sum_{k=1}^M c'_k(\alpha)$
FBNet (Wu et al., 2019)	$\mathcal{L}(w, \alpha) = \mathcal{L}_{\text{task}}(w, \alpha) \cdot \prod_{k=1}^M c_k(\log \alpha)^{\lambda_{\text{hardware}}}$
TAS (Dong & Yang, 2019a)	$\mathcal{L}(w, \alpha) = \mathcal{L}_{\text{task}}(w, \alpha) + (\lambda_{\text{hardware}}/M) \cdot \sum_{k=1}^M h_j(\alpha)$ $h_j(\alpha) = \begin{cases} c'_k(\alpha), & \text{if } c_k(\alpha) > b_k \\ -c'_k(\alpha), & \text{if } c_k(\alpha) < b_k \\ 0, & \text{otherwise} \end{cases}$
TF-NAS (Hu et al., 2020)	$\mathcal{L}(w, \alpha) = \mathcal{L}_{\text{task}}(w, \alpha) + (\lambda_{\text{hardware}}/M) \cdot \sum_{k=1}^M \max\left(\frac{c_k(\alpha)}{b_k} - 1, 0\right)$
HDX (Hong et al., 2022)	$G_\alpha = \begin{cases} \nabla_\alpha \mathcal{L}(w, \alpha), & \text{if } c_k(\alpha) \leq b_k \\ m_\alpha + \nabla_\alpha \mathcal{L}(w, \alpha), & \text{otherwise} \end{cases}$ $\text{where } \mathcal{L}(w, \alpha) = \mathcal{L}_{\text{task}}(w, \alpha) + (\lambda_{\text{hardware}}/M) \cdot \sum_{k=1}^M c'_k(\alpha)$

341  
 342  
 343 distribution sharper and closer to an *argmax*, which promotes exploitation. It is worth noting that  
 344 CONNAS is agnostic to the specific relaxation method used. Other approaches, such as proposed by  
 345 DARTS (Liu et al., 2019) or ProxylessNAS (Cai et al., 2019), although not tested here, could also  
 346 be applied.  
 347

348 We run CONNAS on each search space and corresponding set of hardware constraints for a total of  
 349 150 epochs. During the first 100 epochs, the temperature  $\tau$  is linearly annealed from 10 to 0.1. The  
 350 rescale factor  $R$  is set to 1.2, though additional experiments show that the performance is not very  
 351 sensitive to this choice (see Appendix B). Among the final 50 epochs, we select the architecture  
 352 with the lowest validation loss that satisfies the hardware constraints. Finally, the performance  
 353 of the selected architecture is obtained from the test accuracies provided by NATS-Bench. Each  
 354 experiment is repeated 5 times, as in Dong et al. (2021). The mean and standard deviation of the  
 355 results are reported in Table 2 and Table 3 for the topology and size search spaces, respectively. Since  
 356 NATS-Bench provides performance data for each architecture in the search space, we also report the  
 357 relative error between parentheses to the optimal architecture satisfying the hardware constraints.  
 358 Results of individual runs are shown in Figure 2.

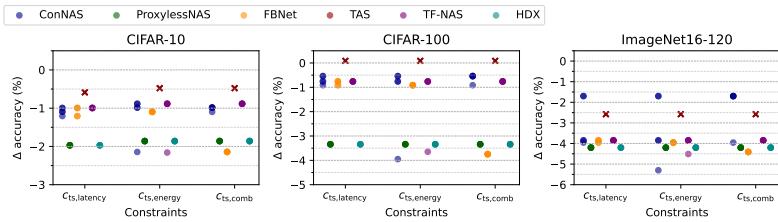
359 We further compare CONNAS against several hardware-aware, gradient-based baselines listed in  
 360 Table 1. We adopt the same experimental setup and training procedure for these baselines to ensure  
 361 a fair comparison. More training details are provided in Appendix E.  
 362

CONNAS consistently identifies architectures that satisfy the specified hardware constraints while  
 363 maintaining strong predictive performance. The resulting architectures exhibit relative errors of  
 364 at most  $-1.18\%$ ,  $-3.38\%$ , and  $-6.45\%$  compared to the optimal feasible solution on CIFAR-  
 365 10, CIFAR-100, and ImageNet16-120, respectively. Compared to the baseline methods using the  
 366 weighting factors listed in Table 1, CONNAS achieves substantially better results than Proxyless-  
 367 NAS and HDX, and a performance comparable to TF-NAS. In contrast, FBNet and TAS do not  
 368 consistently produce valid architectures under all constraint configurations. Additional experiments  
 369 under stricter hardware constraints, presented in Appendix C, further demonstrate the effectiveness  
 370 of CONNAS, which continues to find valid architectures and generally outperforms all baselines.

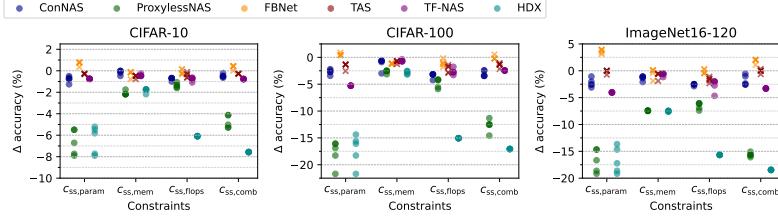
371 It is important to note that the weighting factors used for the baselines may not be optimal under  
 372 the given constraints, and further tuning could potentially improve their performance. Nevertheless,  
 373 this underscores a key advantage of CONNAS: it eliminates the need for extensive hyperparameter  
 374 tuning to balance hardware metrics, thereby eliminating the need to rerun experiments multiple times  
 375 to find an appropriate weighting factor. To illustrate this, we conduct additional experiments with  
 376 alternative weighting factors for the baselines (see Appendix D) and an ablation study on the rescale  
 377 factor  $R$  for CONNAS (see Appendix B). Both analyses confirm that CONNAS is more robust to  
 378 variations in the rescale factor, whereas baseline methods exhibit higher sensitivity to weighting  
 379 factors, requiring careful tuning for each hardware constraint setting.

378 **Table 2: NATS-Bench topology search space classification results.** Test accuracy (mean  $\pm$  std over 5 runs)  
379 is reported under hardware constraints, using on-device measurements of inference latency and energy usage  
380 on an NVIDIA Jetson TX2 Edge GPU. The constraints are chosen so that approximately 50% of architectures  
381 in the search space satisfy the constraints. The relative error compared to the optimal architecture satisfying  
382 the hardware constraints is reported between parentheses. Best results for each constraint and dataset are  
383 highlighted in bold. Performance of runs that do not always satisfy the constraints are reported in gray.  
384

Method	Top-1 accuracy (%)			Runs satisfied
	CIFAR-10	CIFAR-100	ImageNet16-120	
<i>C<sub>ts</sub>,latency</i> : latency $\leq 5.31ms$ (7814 architectures satisfied)				
<b>ConNAS</b>	<b>93.18 <math>\pm</math> 0.09 (-1.08)</b>	<b>70.12 <math>\pm</math> 0.16 (-3.04)</b>	<b>40.59 <math>\pm</math> 1.20 (-6.01)</b>	5 / 5
ProxylessNAS	92.29 $\pm$ 0.00 (-1.97)	67.48 $\pm$ 0.00 (-5.68)	39.40 $\pm$ 0.00 (-7.20)	5 / 5
FBNet	93.18 $\pm$ 0.12 (-1.08)	70.00 $\pm$ 0.08 (-3.16)	39.71 $\pm$ 0.06 (-6.89)	5 / 5
TAS	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
TF-NAS	<b>93.27 <math>\pm</math> 0.00 (-1.00)</b>	70.06 $\pm$ 0.00 (-3.10)	39.75 $\pm$ 0.00 (-6.85)	5 / 5
HDX	75.83 $\pm$ 36.80 (-18.43)	54.18 $\pm$ 29.73 (-18.98)	31.69 $\pm$ 17.25 (-14.91)	5 / 5
<i>C<sub>ts</sub>,energy</i> : energy usage $\leq 23.95mJ$ (7814 architectures satisfied)				
ConNAS	92.97 $\pm$ 0.54 (-1.18)	69.51 $\pm$ 1.48 (-2.43)	<b>40.32 <math>\pm</math> 1.56 (-6.45)</b>	5 / 5
ProxylessNAS	92.29 $\pm$ 0.00 (-1.86)	67.48 $\pm$ 0.00 (-4.46)	39.40 $\pm$ 0.00 (-7.37)	5 / 5
<b>FBNet</b>	<b>93.05 <math>\pm</math> 0.00 (-1.10)</b>	<b>69.91 <math>\pm</math> 0.00 (-2.03)</b>	39.64 $\pm$ 0.00 (-7.12)	5 / 5
TAS	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
TF-NAS	93.01 $\pm$ 0.57 (-1.14)	69.48 $\pm$ 1.29 (-2.46)	39.62 $\pm$ 0.30 (-7.15)	5 / 5
HDX	92.29 $\pm$ 0.00 (-1.86)	67.48 $\pm$ 0.00 (-4.46)	39.40 $\pm$ 0.00 (-7.37)	5 / 5
<i>C<sub>ts</sub>,comb</i> : latency $\leq 5.31ms \wedge$ energy usage $\leq 23.95mJ$ (7586 architectures satisfied)				
<b>ConNAS</b>	93.14 $\pm$ 0.05 (-1.01)	<b>70.21 <math>\pm</math> 0.17 (-1.73)</b>	<b>41.45 <math>\pm</math> 1.01 (-5.08)</b>	5 / 5
ProxylessNAS	92.29 $\pm$ 0.00 (-1.86)	67.48 $\pm$ 0.00 (-4.46)	39.40 $\pm$ 0.00 (-7.13)	5 / 5
FBNet	92.01 $\pm$ 0.00 (-2.14)	67.07 $\pm$ 0.00 (-4.87)	39.19 $\pm$ 0.00 (-7.34)	5 / 5
TAS	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
TF-NAS	<b>93.27 <math>\pm</math> 0.00 (-0.89)</b>	70.06 $\pm$ 0.00 (-1.88)	39.75 $\pm$ 0.00 (-6.78)	5 / 5
HDX	92.29 $\pm$ 0.00 (-1.86)	67.48 $\pm$ 0.00 (-4.46)	39.40 $\pm$ 0.00 (-7.13)	5 / 5



(a) NATS-Bench topology search space



(b) NATS-Bench size search space

425 **Figure 2: Performance comparison of ConNAS against baseline methods.** Performance is measured as  
426 the difference in test accuracy (%) between the architecture found by each method and the optimal architecture  
427 satisfying the hardware constraints. Each point represents a run, runs violating hardware constraints are marked  
428 with a cross.  
429

Table 3: **NATS-Bench size search space classification results.** Test accuracy (mean  $\pm$  std over 5 runs) is reported under hardware constraints, including the number of parameters, number of FLOPs, and peak memory usage. The constraints are chosen such that about 50% of the architectures in each search space meet them. The relative error compared to the optimal architecture satisfying the hardware constraints is reported between parentheses. Best results for each constraint and dataset are highlighted in bold. Performance of runs that do not always satisfy the constraints are reported in gray.

Method	Top-1 accuracy (%)			Runs satisfied
	CIFAR-10	CIFAR-100	ImageNet16-120	
<i>C<sub>ss, param</sub> : #parameters <math>\leq 261650</math> (16385 architectures satisfied)</i>				
ConNAS	$91.74 \pm 0.29$ (-.66)	$66.22 \pm 0.47$ (-.27)	$39.77 \pm 0.77$ (-.22)	5 / 5
ProxylessNAS	$85.87 \pm 1.16$ (-.66)	$51.11 \pm 2.35$ (-.17)	$25.27 \pm 2.13$ (-.16)	5 / 5
FBNet	$93.22 \pm 0.20$	$69.43 \pm 0.22$	$45.68 \pm 0.37$	0 / 5
TAS	$92.25 \pm 0.02$	$67.34 \pm 0.54$	$41.89 \pm 0.33$	0 / 5
TF-NAS	$91.78 \pm 0.00$ (-.75)	$63.64 \pm 0.00$ (-.58)	$38.00 \pm 0.00$ (-.43)	5 / 5
HDX	$86.10 \pm 1.28$ (-.43)	$51.71 \pm 2.88$ (-.17)	$25.36 \pm 2.42$ (-.16)	5 / 5
<i>C<sub>ss, mem</sub> : peak mem. <math>\leq 655kB</math> (20480 architectures satisfied)</i>				
ConNAS	$93.28 \pm 0.15$ (-.14)	$69.70 \pm 1.02$ (-.16)	$44.58 \pm 0.43$ (-.13)	5 / 5
ProxylessNAS	$91.33 \pm 0.19$ (-.20)	$68.22 \pm 0.28$ (-.24)	$38.49 \pm 0.03$ (-.74)	5 / 5
FBNet	$93.16 \pm 0.29$	$69.69 \pm 0.05$	$45.53 \pm 0.87$	1 / 5
TAS	$92.93 \pm 0.19$ (-.49)	$70.10 \pm 0.32$ (-.76)	$44.98 \pm 0.59$ (-.95)	5 / 5
TF-NAS	$92.99 \pm 0.10$ (-.43)	$70.21 \pm 0.16$ (-.65)	$45.25 \pm 0.27$ (-.69)	5 / 5
HDX	$91.59 \pm 0.19$ (-.83)	$68.03 \pm 0.29$ (-.28)	$38.41 \pm 0.07$ (-.73)	5 / 5
<i>C<sub>ss, flops</sub> : #FLOPs <math>\leq 344194M</math> (16385 architectures satisfied)</i>				
ConNAS	$92.22 \pm 0.14$ (-.75)	$67.10 \pm 0.48$ (-.38)	$41.79 \pm 0.16$ (-.28)	5 / 5
ProxylessNAS	$91.59 \pm 0.21$ (-.38)	$65.68 \pm 0.85$ (-.80)	$37.84 \pm 0.62$ (-.53)	5 / 5
FBNet	$92.87 \pm 0.23$	$69.42 \pm 0.60$	$44.23 \pm 0.39$	0 / 5
TAS	$92.57 \pm 0.23$	$68.36 \pm 0.68$	$42.71 \pm 0.46$	0 / 5
TF-NAS	$92.19 \pm 0.18$ (-.78)	$67.81 \pm 0.56$ (-.27)	$41.69 \pm 1.16$ (-.28)	5 / 5
HDX	$86.88 \pm 0.00$ (-.69)	$55.42 \pm 0.00$ (-.15)	$28.70 \pm 0.00$ (-.15)	5 / 5
<i>C<sub>ss, comb</sub> : #parameters <math>\leq 261650 \wedge</math> peak mem. <math>\leq 655kB \wedge</math> #FLOPs <math>\leq 344194M</math> (13342 architectures satisfied)</i>				
ConNAS	$92.05 \pm 0.16$ (-.48)	$65.87 \pm 0.55$ (-.04)	$40.22 \pm 0.96$ (-.11)	5 / 5
ProxylessNAS	$87.76 \pm 0.59$ (-.47)	$56.22 \pm 1.18$ (-.12)	$26.39 \pm 0.37$ (-.15)	5 / 5
FBNet	$92.88 \pm 0.19$	$68.83 \pm 0.34$	$43.88 \pm 0.42$	0 / 5
TAS	$92.26 \pm 0.01$	$67.47 \pm 0.42$	$41.97 \pm 0.35$	0 / 5
TF-NAS	$91.76 \pm 0.00$ (-.77)	$66.46 \pm 0.00$ (-.24)	$38.73 \pm 0.00$ (-.33)	5 / 5
HDX	$84.96 \pm 0.00$ (-.75)	$51.84 \pm 0.00$ (-.17)	$23.57 \pm 0.00$ (-.18)	5 / 5

## 4.2 PRACTICAL USE CASE: CONDITION MONITORING OF INDUCTION MOTORS

Condition monitoring of industrial assets has received an increase interest over the years (Surucu et al., 2023), preventing unexpected failures and costly downtimes. By leveraging edge machine learning, data from these assets can be captured and processed locally to detect faults in real-time. To demonstrate the applicability of our work, we validate our approach through a use case focused on condition monitoring of induction motors, aiming to detect eccentricity faults using a neural network deployed on an edge device. An eccentricity fault occurs when the rotor is not perfectly centered within the stator, resulting in an uneven air gap, causing an unbalanced magnetic pull (Desenfans et al., 2024). The prediction task involves classifying the type of eccentricity, specifically distinguishing between no eccentricity fault, static eccentricity (where the rotor remains consistently off-center relative to the stator), dynamic eccentricity (where the rotor's off-center position rotates over time relative to the stator), and mixed eccentricity (a combination of static and dynamic eccentricity).

The goal in this use case is to discover neural network architectures that are deployable across a wide range of microcontrollers with diverse hardware capabilities. To define realistic deployment constraints, we base our hardware constraints on 6 hand-crafted architectures (listed in Appendix G), each having different resource requirements. We construct a search space based on 1D convolutions consisting of 8 layers. Each layer has varying configurations in terms of the number of filters, convolution types, and the inclusion of skip connections, resulting in a total of 1.94 billion unique

486 architectures. The detailed description of the search space is provided in Appendix F. The dataset  
 487 contains 31,920 instances, split into 25,530 for training and 6,390 for testing. Each instance consists  
 488 of 256 time steps sampled at 5 kHz, with eight features: *stator currents* (3), *phase voltages* (3),  
 489 *rotor speed* (1), and *rotor angle* (1). We use 80% of the training set to train the supernet, while  
 490 the remaining 20% serves as a validation set for architecture selection. The training and selection  
 491 procedure follows the same approach outlined in Section 4.1. The selected architectures are retrained  
 492 from scratch using five-fold cross-validation over 200 epochs. Their performance is subsequently  
 493 evaluated on the test set. We compare the best architectures found by CONNas with hand-crafted  
 494 models<sup>1</sup>.

495 **Table 4: Classification performance and resource usage of models discovered by CONNas under various**  
 496 **hardware constraints, compared to handcrafted baseline models.** The constraints  $c_1$ ,  $c_2$ ,  $c_3$ , and  $c_4$  are  
 497 defined as follows:  $c_1$ : model size  $\leq$  2MB, peak memory usage  $\leq$  640kB;  $c_2$ : model size  $\leq$  256kB, peak  
 498 memory usage  $\leq$  112kB;  $c_3$ : model size  $\leq$  128kB, peak memory usage  $\leq$  36kB;  $c_4$ : model size  $\leq$  64kB, peak  
 499 memory usage  $\leq$  18kB.

Name	Top-1 accuracy (%)	Satisfied constraints				Model size (bytes)	Peak memory usage (bytes)
		$c_1$	$c_2$	$c_3$	$c_4$		
ConNAS (unconstrained)	97.36 $\pm$ 0.27					6.06M $\pm$ 1.61M	524k $\pm$ 0.00
ConNAS + $c_1$	97.09 $\pm$ 0.47	✓				1.93M $\pm$ 13.2k	524k $\pm$ 0.00
ConNAS + $c_2$	96.17 $\pm$ 0.84	✓	✓			122k $\pm$ 30.7k	69.6k $\pm$ 0.00k
ConNAS + $c_3$	94.96 $\pm$ 0.69	✓	✓	✓		26.0k $\pm$ 13.5k	16.8k $\pm$ 10.1k
ConNAS + $c_4$	94.37 $\pm$ 1.94	✓	✓	✓	✓	8.69k $\pm$ 2.12k	12.3k $\pm$ 0.00
Conv1D-Reg	95.54 $\pm$ 0.29	✓				135k	16.4k
Conv1D-DS	93.07 $\pm$ 0.35	✓	✓	✓	✓	51.3k	16.4k
Conv1D-Reg-Max	93.31 $\pm$ 1.86					22.1M	524k
Conv1D-DS-Max	96.72 $\pm$ 0.24					7.50M	524k
Conv1D-Reg-Min	90.15 $\pm$ 4.33	✓	✓	✓	✓	3.47k	12.3k
Conv1D-DS-Min	93.93 $\pm$ 1.64	✓	✓	✓	✓	2.32k	12.3k

513 Table 4 compares the performance and resource consumption of architectures discovered by CON-  
 514 NAS with manually designed baselines. CONNas consistently identifies architectures that outper-  
 515 form handcrafted ones in terms of accuracy under specific hardware constraints. For example, under  
 516 constraint  $c_1$ , CONNas discovers architectures that achieve an accuracy of 97.07%, which is 1.55%  
 517 higher than Conv1D-Reg, the best manually designed model under the same constraint. At  $c_4$ , the  
 518 discovered architecture reaches 94.37% accuracy, compared to 93.93% for Conv1D-Reg-Min.

## 5 CONCLUSION

522 We introduced CONNas, a novel hardware-aware, gradient-based NAS technique that explicitly en-  
 523 forces hardware constraints during the search process. Unlike prior approaches that rely on weighted  
 524 loss terms for hardware-aware regularization, CONNas directly modifies the gradients of archi-  
 525 tecture parameters when hardware constraints are violated, effectively steering the search toward  
 526 hardware-feasible solutions. Experiments on NATS-Bench demonstrate that CONNas consistently  
 527 discovers architectures that satisfy various hardware constraints, achieving performance close to the  
 528 optimal and overall outperforming existing baseline methods in both performance and compliance  
 529 with hardware constraints. Furthermore, we validated CONNas in a practical use case, where it  
 530 successfully identified high-performing architectures under tight resource limitations. These results  
 531 highlight CONNas as a promising approach for the automatic design of deep learning models for  
 532 deployment in resource-constrained environments.

533  
 534  
 535  
 536  
 537  
 538 <sup>1</sup>We could have also compared to the baseline methods as in Section 4.1. However, since all baseline  
 539 methods require differentiable hardware metrics, this comparison is not feasible as factorizing the metrics with  
 respect to  $\alpha$  is intractable due to the large search space.

540 REFERENCES  
541

542 Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network archi-  
543     tectures using reinforcement learning. In *International Conference on Learning Representations*,  
544     2017.

545 Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and  
546     Quoc V. Le. Can weight sharing outperform random architecture search? an investigation with  
547     TuNAS. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recog-*  
548     *nition*, pp. 14323–14332, 2020.

549 Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target  
550     task and hardware. In *International Conference on Learning Representations*, 2019.

551 Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff,  
552     Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image  
553     prediction. In *Advances in Neural Information Processing Systems*, volume 31. Curran Asso-  
554     ciates, Inc., 2018.

555 Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings*  
556     *of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1251–1258, 2017.

557 Philip Desenfans, Zifeng Gong, Dries Vanoost, Konstantinos Gryllias, Jeroen Boydens, and Davy  
558     Pissoort. The influence of the unbalanced magnetic pull on fault-induced rotor eccentricity in  
559     induction motors. *Journal of Vibration and Control*, 30(5):943–959, 2024.

560 Philip Desenfans, Zifeng Gong, Dries Vanoost, and Davy Pissoort. A python magnetic  
561     equivalent circuit model for rapid data generation of induction motors in faulty condi-  
562     tions., 2025. URL [https://gitlab.kuleuven.be/m-group-campus-brugge/wavecore\\_public/phd-philip-desenfans/imme](https://gitlab.kuleuven.be/m-group-campus-brugge/wavecore_public/phd-philip-desenfans/imme).

563 Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Advances in*  
564     *Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019a.

565 Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *Proceed-*  
566     *ings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1761–1770,  
567     2019b.

568 Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. NATS-bench: Benchmarking NAS  
569     algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Ma-*  
570     *chine Intelligence*, pp. 1–1, 2021. ISSN 0162-8828, 2160-9292, 1939-3539.

571 Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architec-  
572     ture search via lamarckian evolution. In *International Conference on Learning Representations*,  
573     2019a.

574 Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey.  
575     *Journal of Machine Learning Research*, 20(55):1–21, 2019b.

576 Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly,  
577     Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep  
578     neural networks for acoustic modeling in speech recognition: The shared views of four research  
579     groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012. ISSN 1558-0792.

580 Deokki Hong, Kanghyun Choi, Hye Yoon Lee, Joonsang Yu, Noseong Park, Youngsok Kim,  
581     and Jinho Lee. Enabling hard constraints in differentiable neural network and accelerator co-  
582     exploration. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC ’22, pp.  
583     589–594. Association for Computing Machinery, 2022. ISBN 978-1-4503-9142-9. doi: 10.1145/  
584     3489517.3530507. URL <https://dl.acm.org/doi/10.1145/3489517.3530507>.

585 Yibo Hu, Xiang Wu, and Ran He. TF-NAS: Rethinking three search freedoms of latency-constrained  
586     differentiable neural architecture search. In *In European conference on computer vision*, pp. 123–  
587     139. Cham: Springer International Publishing, 2020.

594 Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In  
 595 *International Conference on Learning Representations*, 2017.

596

597 Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep con-  
 598 volutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. ISSN 0001-0782,  
 599 1557-7317.

600 Colin Lea, Michael D. Flynn, Rene Vidal, Austin Reiter, and Gregory D. Hager. Temporal convo-  
 601 lutional networks for action segmentation and detection. In *Proceedings of the IEEE Conference*  
 602 *on Computer Vision and Pattern Recognition*, pp. 156–165, 2017.

603

604 Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue  
 605 Wang, Cong Hao, and Yingyan Lin. HW-NAS-bench: Hardware-aware neural architecture search  
 606 benchmark. 2021.

607

608 Edgar Liberis, Łukasz Dudziak, and Nicholas D. Lane.  $\mu$ nas: Constrained neural architecture search  
 609 for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*,  
 610 EuroMLSys ’21, pp. 70–79, New York, NY, USA, 2021. Association for Computing Machinery.  
 ISBN 9781450382984.

611

612 Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. MCUNet: Tiny deep  
 613 learning on IoT devices. In *Advances in Neural Information Processing Systems*, volume 33, pp.  
 11711–11722. Curran Associates, Inc., 2020.

614

615 Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In  
 616 *International Conference on Learning Representations (2019)*, 2019.

617

618 Niv Nayman, Yonathan Aflalo, Asaf Noy, and Lihi Zelnik. Hardcore-nas: Hard constrained dif-  
 619 ferentiable neural architecture search. In Marina Meila and Tong Zhang (eds.), *Proceedings of*  
 620 *the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine*  
 621 *Learning Research*, pp. 7979–7990. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/nayman21a.html>.

622

623 NVIDIA Inc. NVIDIA jetson TX2: High performance AI at the edge. URL <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/>.

624

625 Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan,  
 626 Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of*  
 627 *the 34th International Conference on Machine Learning - Volume 70*, ICML’17, pp. 2902–2911.  
 628 JMLR.org, 2017.

629

630 Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a  
 631 scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

632

633 Onur Surucu, Stephen Andrew Gadsden, and John Yawney. Condition monitoring using machine  
 634 learning: A review of theory, applications, and recent advances. *Expert Systems with Applications*,  
 221:119738, 2023. ISSN 0957-4174.

635

636 Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and  
 637 Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of*  
 638 *the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2820–2828, 2019.

639

640 Quinten Van Baelen and Peter Karsmakers. Constraint guided gradient descent: Training with in-  
 641 equality constraints with applications in regression and semantic segmentation. *Neurocomputing*,  
 556:126636, 2023. ISSN 0925-2312.

642

643 Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu,  
 644 Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, and Joseph E. Gonzalez. FBNetV2: Differentiable  
 645 neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF*  
 646 *Conference on Computer Vision and Pattern Recognition*, pp. 12962–12971, 2020.

647

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural  
 architectures for neural architecture search. In *AAAI*, pp. 10293–10301, 2021.

648 Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian,  
649 Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient ConvNet design  
650 via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on*  
651 *Computer Vision and Pattern Recognition*, pp. 10726–10734, 2019.

652 Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search.  
653 In *International Conference on Learning Representations*, 2019.

654 Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arik, Haonan Yu, Hairong Liu, and Greg Diamos.  
655 Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912*, 2018.

656 Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International*  
657 *Conference on Learning Representations*, 2017.

658 Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures  
659 for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and*  
660 *Pattern Recognition*, pp. 8697–8710, 2018.

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

---

702 **A COMPUTATION HARDWARE METRICS**  
703

704 In this work, we focus on three hardware metrics: the number of parameters,  
705 the number of FLOPs, and the peak memory usage during inference. The following  
706 subsections describe how each of these metrics is computed. It is important to note  
707 that our proposed method is not limited to these metrics and can be applied to any  
708 hardware metric, whether derived analytically, measured through profiling,  
709 or predicted by learned models.

710 **A.1 NUMBER OF PARAMETERS**  
711

712 The total number of parameters in a candidate architecture is computed by summing the number  
713 of parameters of each individual layer. To estimate the model size in bytes, we multiply the total  
714 number of parameters by 4, assuming 32-bit floating-point representation.

715 **A.1.1 CONVOLUTIONAL BLOCK**  
716

717 A convolutional block consists of a convolutional layer, followed by a batch normalization and a  
718 ReLU activation. We assume that batch normalization is fused into the convolutional layer during  
719 inference.

720 *Regular Convolution:*

721 
$$\text{params} = (C_{in} \times k_h \times k_w + 1) \times C_{out} \quad (8)$$
722

723 where  $C_{in}$  and  $C_{out}$  are the input and output channels,  $k_h$  and  $k_w$  are kernel dimensions, and the  
724 +1 accounts for the bias term.

725 *Depthwise Separable Convolution:*

726 
$$\text{params} = (C_{in} \times k_h \times k_w + 1) + (C_{in} \times C_{out} + 1) \quad (9)$$
727

728 where the first term corresponds to the depthwise convolution and the second to the pointwise  
729 convolution.

730 **A.2 FULLY CONNECTED LAYER**  
731

732 
$$\text{params} = (N_{in} + 1) \times N_{out} \quad (10)$$
733

734 where  $N_{in}$  and  $N_{out}$  are the input and output features, respectively, and the +1 accounts for the bias  
735 term.

736 **A.3 NUMBER OF FLOPS**  
737

738 Similar to the parameter count, the total number of FLOPs is computed by summing the FLOPs of  
739 each individual layer.

740 **A.3.1 CONVOLUTIONAL LAYER**  
741

742 
$$\text{FLOPs} = 2 \times (C_{in} \times k_h \times k_w + 1) \times H_{out} \times W_{out} \times C_{out} \quad (11)$$
743

744 Here,  $H_{out}$  and  $W_{out}$  are the output feature map dimensions.

745 **A.3.2 RELU**  
746

747 
$$\text{FLOPs} = H_{out} \times W_{out} \times C_{out} \quad (12)$$
748

749 **A.3.3 LINEAR LAYER**  
750

751 
$$\text{FLOPs} = 2 \times (N_{in} + 1) \times N_{out} + N_{out} \quad (13)$$
752

753 **A.3.4 GLOBAL POOLING LAYER**  
754

755 
$$\text{FLOPs} = (k_h \times k_w + 1) \times H_{out} \times W_{out} \times C_{out} \quad (14)$$

756 A.4 PEAK MEMORY USAGE  
757758 Peak memory usage during inference is estimated by computing the combined size of input and  
759 output feature maps for each layer. The maximum of these values across all layers is taken as the  
760 peak memory usage.  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809

810  
811  
**B ABLATION RESCALE FACTOR**  
812813  
814  
815  
816  
817  
**Table 5: NATS-Bench topology search space classification results on ConNAS with different rescale  
factors.** Test accuracy (mean  $\pm$  std over 5 runs) is reported under hardware constraints, using on-device mea-  
surements of inference latency and energy usage on an NVIDIA Jetson TX2 Edge GPU. The constraints are  
chosen so that approximately 50% of architectures in the search space satisfy the constraints. The relative error  
compared to the optimal architecture satisfying the hardware constraints is reported between parentheses.

818 819 <b>Rescaling factor</b>	820 <b>Top-1 accuracy (%)</b>			821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863
	820 <b>CIFAR-10</b>	820 <b>CIFAR-100</b>	820 <b>ImageNet16-120</b>	820 <b>Runs satisfied</b>
<i>c<sub>ts, latency</sub> : latency <math>\leq 5.31ms</math> (7814 architectures satisfied)</i>				
1.2	93.18 $\pm$ 0.09 (-1.08)	70.12 $\pm$ 0.16 (-3.04)	40.59 $\pm$ 1.20 (-6.01)	5 / 5
1.5	93.20 $\pm$ 0.09 (-1.06)	70.07 $\pm$ 0.13 (-3.09)	40.16 $\pm$ 0.97 (-6.44)	5 / 5
2.0	93.21 $\pm$ 0.05 (-1.05)	70.19 $\pm$ 0.12 (-2.97)	41.04 $\pm$ 1.18 (-5.56)	5 / 5
5.0	93.25 $\pm$ 0.04 (-1.02)	70.10 $\pm$ 0.10 (-3.06)	40.18 $\pm$ 0.96 (-6.42)	5 / 5
10.0	92.74 $\pm$ 0.67 (-1.52)	68.83 $\pm$ 1.79 (-4.33)	39.60 $\pm$ 1.48 (-7.00)	5 / 5
<i>c<sub>ts, energy</sub> : energy usage <math>\leq 23.95mJ</math> (7814 architectures satisfied)</i>				
1.2	92.97 $\pm$ 0.54 (-1.18)	69.51 $\pm$ 1.48 (-2.43)	40.32 $\pm$ 1.56 (-6.45)	5 / 5
1.5	93.20 $\pm$ 0.09 (-0.95)	70.07 $\pm$ 0.13 (-1.87)	40.16 $\pm$ 0.97 (-6.61)	5 / 5
2.0	92.95 $\pm$ 0.53 (-1.20)	69.55 $\pm$ 1.50 (-2.39)	40.75 $\pm$ 1.66 (-6.02)	5 / 5
5.0	93.17 $\pm$ 0.00 (-0.98)	70.28 $\pm$ 0.00 (-1.66)	41.90 $\pm$ 0.00 (-4.87)	5 / 5
10.0	92.72 $\pm$ 0.65 (-1.43)	68.87 $\pm$ 1.83 (-3.07)	40.03 $\pm$ 1.81 (-6.74)	5 / 5
<i>c<sub>ts, comb</sub> : latency <math>\leq 5.31ms \wedge</math> energy usage <math>\leq 23.95mJ</math> (7586 architectures satisfied)</i>				
1.2	93.14 $\pm$ 0.05 (-1.01)	70.21 $\pm$ 0.17 (-1.73)	41.45 $\pm$ 1.01 (-5.08)	5 / 5
1.5	92.95 $\pm$ 0.53 (-1.20)	69.55 $\pm$ 1.50 (-2.39)	40.75 $\pm$ 1.66 (-5.78)	5 / 5
2.0	93.17 $\pm$ 0.00 (-0.98)	70.28 $\pm$ 0.00 (-1.66)	41.90 $\pm$ 0.00 (-4.63)	5 / 5
5.0	92.89 $\pm$ 0.50 (-1.26)	69.45 $\pm$ 1.45 (-2.49)	40.28 $\pm$ 1.58 (-6.25)	5 / 5
10.0	92.49 $\pm$ 0.66 (-1.66)	68.19 $\pm$ 1.81 (-3.75)	39.31 $\pm$ 1.58 (-7.22)	5 / 5

864 **Table 6: NATS-Bench size search space classification results on ConNAS with different rescale factors.**  
 865 Test accuracy (mean  $\pm$  std over 5 runs) is reported under hardware constraints, including the number of  
 866 parameters, number of FLOPs, and peak memory usage. The constraints are chosen so that about 50% of the  
 867 architectures in each search space meet them. The relative error compared to the optimal architecture satisfying  
 868 the hardware constraints is reported between parentheses.

Rescaling factor	Top-1 accuracy (%)			Runs satisfied
	CIFAR-10	CIFAR-100	ImageNet16-120	
$c_{ss, \text{param}} : \#\text{parameters} \leq 261650$ (16385 architectures satisfied)				
1.2	91.74 $\pm$ 0.29 (-0.79)	66.22 $\pm$ 0.47 (-2.70)	39.77 $\pm$ 0.77 (-2.27)	5 / 5
1.5	92.09 $\pm$ 0.24 (-0.44)	65.92 $\pm$ 0.65 (-3.00)	40.29 $\pm$ 1.10 (-1.74)	5 / 5
2.0	91.87 $\pm$ 0.07 (-0.66)	66.22 $\pm$ 0.60 (-2.70)	39.48 $\pm$ 1.29 (-2.55)	5 / 5
5.0	91.85 $\pm$ 0.19 (-0.68)	65.97 $\pm$ 0.38 (-2.95)	39.55 $\pm$ 0.53 (-2.48)	5 / 5
10.0	91.67 $\pm$ 0.08 (-0.86)	65.99 $\pm$ 0.36 (-2.93)	39.31 $\pm$ 0.21 (-2.73)	5 / 5
$c_{ss, \text{mem}} : \text{peak mem.} \leq 655kB$ (20480 architectures satisfied)				
1.2	93.28 $\pm$ 0.20 (-0.14)	69.70 $\pm$ 1.02 (-1.16)	44.58 $\pm$ 0.43 (-1.35)	5 / 5
1.5	93.26 $\pm$ 0.15 (-0.16)	70.14 $\pm$ 0.32 (-0.72)	44.67 $\pm$ 0.68 (-1.26)	5 / 5
2.0	93.19 $\pm$ 0.17 (-0.23)	69.54 $\pm$ 0.95 (-1.32)	44.42 $\pm$ 0.38 (-1.51)	5 / 5
5.0	93.26 $\pm$ 0.16 (-0.16)	70.06 $\pm$ 0.15 (-0.80)	44.65 $\pm$ 0.18 (-1.29)	5 / 5
10.0	93.29 $\pm$ 0.11 (-0.13)	70.01 $\pm$ 0.24 (-0.85)	44.62 $\pm$ 0.22 (-1.31)	5 / 5
$c_{ss, \text{flops}} : \#\text{FLOPs} \leq 344194M$ (16385 architectures satisfied)				
1.2	92.22 $\pm$ 0.14 (-0.75)	67.10 $\pm$ 0.48 (-3.38)	41.79 $\pm$ 0.16 (-2.57)	5 / 5
1.5	92.23 $\pm$ 0.19 (-0.74)	67.00 $\pm$ 0.80 (-3.48)	41.45 $\pm$ 0.92 (-2.92)	5 / 5
2.0	92.10 $\pm$ 0.16 (-0.87)	67.08 $\pm$ 1.24 (-3.40)	41.63 $\pm$ 1.19 (-2.74)	5 / 5
5.0	92.22 $\pm$ 0.08 (-0.75)	67.54 $\pm$ 1.09 (-2.94)	42.09 $\pm$ 0.53 (-2.28)	5 / 5
10.0	92.31 $\pm$ 0.14 (-0.66)	66.76 $\pm$ 0.73 (-3.72)	41.70 $\pm$ 0.51 (-2.67)	5 / 5
$c_{ss, \text{comb}} : \#\text{parameters} \leq 261650 \wedge \text{peak mem.} \leq 655kB \wedge \#\text{FLOPs} \leq 344194M$ (13342 architectures satisfied)				
1.2	92.01 $\pm$ 0.19 (-0.52)	66.65 $\pm$ 0.39 (-2.27)	41.01 $\pm$ 0.33 (-1.02)	5 / 5
1.5	91.72 $\pm$ 0.16 (-0.81)	66.02 $\pm$ 0.70 (-2.90)	40.23 $\pm$ 0.51 (-1.81)	5 / 5
2.0	91.99 $\pm$ 0.38 (-0.54)	66.15 $\pm$ 0.45 (-2.77)	40.63 $\pm$ 0.98 (-1.40)	5 / 5
5.0	91.94 $\pm$ 0.27 (-0.59)	66.02 $\pm$ 0.50 (-2.90)	40.57 $\pm$ 0.84 (-1.46)	5 / 5
10.0	91.67 $\pm$ 0.36 (-0.86)	65.36 $\pm$ 0.89 (-3.56)	40.13 $\pm$ 0.65 (-1.91)	5 / 5

893  
 894  
 895  
 896  
 897  
 898  
 899  
 900  
 901  
 902  
 903  
 904  
 905  
 906  
 907  
 908  
 909  
 910  
 911  
 912  
 913  
 914  
 915  
 916  
 917

918 C EXPERIMENTS WITH STRICTER HARDWARE CONSTRAINTS  
919920  
921 **Table 7: NATS-Bench topology search space classification results under stricter hardware constraints.**  
922 Test accuracy (mean  $\pm$  std over 5 runs) is reported under hardware constraints, using on-device measurements  
923 of inference latency and energy usage on an NVIDIA Jetson TX2 Edge GPU. Stricter hardware constraints were  
924 set so that approximately 25% and 10% of architectures in the search space satisfy the constraints. The relative  
925 error compared to the optimal architecture satisfying the hardware constraints is reported between parentheses.  
926

Method	Top-1 accuracy (%)			Runs satisfied
	CIFAR-10	CIFAR-100	ImageNet16-120	
<i>c<sub>ts,latency25</sub> : latency <math>\leq 4.14ms</math> (3908 architectures satisfied)</i>				
ConNAS	92.08 $\pm$ 0.07 (-1.97)	67.09 $\pm$ 0.20 (-4.43)	39.09 $\pm$ 0.72 (-5.88)	5 / 5
<b>ProxylessNAS</b>	<b>92.29 <math>\pm</math> 0.00 (-1.76)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.04)</b>	<b>39.40 <math>\pm</math> 0.00 (-5.57)</b>	5 / 5
FBNet	93.22 $\pm$ 0.09	70.03 $\pm$ 0.07	39.73 $\pm$ 0.05	0 / 5
TAS	93.27 $\pm$ 0.00	70.06 $\pm$ 0.00	39.75 $\pm$ 0.00	0 / 5
<b>TF-NAS</b>	<b>92.29 <math>\pm</math> 0.00 (-1.76)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.04)</b>	<b>39.40 <math>\pm</math> 0.00 (-5.57)</b>	5 / 5
HDX	10.00 $\pm$ 0.00 (-84.05)	1.00 $\pm$ 0.00 (-70.52)	0.83 $\pm$ 0.00 (-44.13)	5 / 5
<i>c<sub>ts,latency10</sub> : latency <math>\leq 3.03ms</math> (1564 architectures satisfied)</i>				
<b>ConNAS</b>	<b>92.05 <math>\pm</math> 0.07 (-0.73)</b>	<b>67.01 <math>\pm</math> 0.20 (-1.35)</b>	<b>38.83 <math>\pm</math> 0.72 (-1.61)</b>	5 / 5
ProxylessNAS	92.29 $\pm$ 0.00	67.48 $\pm$ 0.00	39.40 $\pm$ 0.00	0 / 5
FBNet	93.22 $\pm$ 0.09	70.03 $\pm$ 0.07	39.73 $\pm$ 0.05	0 / 5
TAS	92.00 $\pm$ 0.01	67.13 $\pm$ 0.05	39.13 $\pm$ 0.05	0 / 5
<b>TF-NAS</b>	10.00 $\pm$ 0.00 (-82.78)	1.00 $\pm$ 0.00 (-67.36)	0.83 $\pm$ 0.00 (-39.60)	5 / 5
HDX	10.00 $\pm$ 0.00 (-82.78)	1.00 $\pm$ 0.00 (-67.36)	0.83 $\pm$ 0.00 (-39.60)	5 / 5
<i>c<sub>ts,energy25</sub> : energy usage <math>\leq 18.6mJ</math> (3908 architectures satisfied)</i>				
ConNAS	92.06 $\pm$ 0.06 (-1.98)	67.09 $\pm$ 0.20 (-4.43)	39.04 $\pm$ 0.68 (-5.93)	5 / 5
<b>ProxylessNAS</b>	<b>92.29 <math>\pm</math> 0.00 (-1.76)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.04)</b>	<b>39.40 <math>\pm</math> 0.00 (-5.57)</b>	5 / 5
FBNet	93.14 $\pm$ 0.12	69.97 $\pm$ 0.08	39.69 $\pm$ 0.06	0 / 5
TAS	93.01 $\pm$ 0.57	69.48 $\pm$ 1.29	39.62 $\pm$ 0.30	1 / 5
<b>TF-NAS</b>	<b>92.29 <math>\pm</math> 0.00 (-1.76)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.04)</b>	<b>39.40 <math>\pm</math> 0.00 (-5.57)</b>	5 / 5
HDX	10.00 $\pm$ 0.00 (-84.05)	1.00 $\pm$ 0.00 (-70.52)	0.83 $\pm$ 0.00 (-44.13)	5 / 5
<i>c<sub>ts,energy10</sub> : energy usage <math>\leq 13.3mJ</math> (1564 architectures satisfied)</i>				
ConNAS	92.05 $\pm$ 0.07 (-0.73)	67.01 $\pm$ 0.20 (-1.35)	38.83 $\pm$ 0.72 (-1.61)	5 / 5
<b>ProxylessNAS</b>	<b>92.29 <math>\pm</math> 0.00 (-0.49)</b>	<b>67.48 <math>\pm</math> 0.00 (-0.88)</b>	<b>39.40 <math>\pm</math> 0.00 (-1.03)</b>	5 / 5
FBNet	93.14 $\pm$ 0.12	69.97 $\pm$ 0.08	39.69 $\pm$ 0.06	0 / 5
TAS	92.00 $\pm$ 0.01	67.11 $\pm$ 0.05	39.15 $\pm$ 0.05	3 / 5
<b>TF-NAS</b>	10.00 $\pm$ 0.00 (-82.78)	1.00 $\pm$ 0.00 (-67.36)	0.83 $\pm$ 0.00 (-39.60)	5 / 5
HDX	10.00 $\pm$ 0.00 (-82.78)	1.00 $\pm$ 0.00 (-67.36)	0.83 $\pm$ 0.00 (-39.60)	5 / 5
<i>c<sub>ts,comb25</sub> : latency <math>\leq 4.14ms \wedge</math> energy usage <math>\leq 18.6mJ</math> (3785 architectures satisfied)</i>				
ConNAS	92.10 $\pm$ 0.06 (-1.95)	67.16 $\pm$ 0.16 (-4.36)	39.35 $\pm$ 0.59 (-5.61)	5 / 5
<b>ProxylessNAS</b>	<b>92.29 <math>\pm</math> 0.00 (-1.76)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.04)</b>	<b>39.40 <math>\pm</math> 0.00 (-5.57)</b>	5 / 5
FBNet	92.01 $\pm$ 0.00 (-2.04)	67.07 $\pm$ 0.00 (-4.45)	39.19 $\pm$ 0.00 (-5.78)	5 / 5
TAS	93.27 $\pm$ 0.00	70.06 $\pm$ 0.00	39.75 $\pm$ 0.00	0 / 5
<b>TF-NAS</b>	<b>92.29 <math>\pm</math> 0.00 (-1.76)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.04)</b>	<b>39.40 <math>\pm</math> 0.00 (-5.57)</b>	5 / 5
HDX	10.00 $\pm$ 0.00 (-84.05)	1.00 $\pm$ 0.00 (-70.52)	0.83 $\pm$ 0.00 (-44.13)	5 / 5
<i>c<sub>ts,comb10</sub> : latency <math>\leq 3.03ms \wedge</math> energy usage <math>\leq 13.3mJ</math> (1526 architectures satisfied)</i>				
<b>ConNAS</b>	<b>92.08 <math>\pm</math> 0.06 (-0.70)</b>	<b>67.20 <math>\pm</math> 0.21 (-1.16)</b>	<b>39.10 <math>\pm</math> 0.70 (-1.33)</b>	5 / 5
ProxylessNAS	92.29 $\pm$ 0.00	67.48 $\pm$ 0.00	39.40 $\pm$ 0.00	0 / 5
FBNet	92.01 $\pm$ 0.00	67.07 $\pm$ 0.00	39.19 $\pm$ 0.00	0 / 5
TAS	91.99 $\pm$ 0.01	67.15 $\pm$ 0.04	39.11 $\pm$ 0.04	0 / 5
<b>TF-NAS</b>	10.00 $\pm$ 0.00 (-82.78)	1.00 $\pm$ 0.00 (-67.36)	0.83 $\pm$ 0.00 (-39.60)	5 / 5
HDX	10.00 $\pm$ 0.00 (-82.78)	1.00 $\pm$ 0.00 (-67.36)	0.83 $\pm$ 0.00 (-39.60)	5 / 5

964  
965  
966  
967  
968  
969  
970  
971

972 **Table 8: NATS-Bench size search space classification results under stricter hardware constraints.** Test  
973 accuracy (mean  $\pm$  std over 5 runs) is reported under hardware constraints, including the number of parameters,  
974 number of FLOPs, and peak memory usage. Stricter hardware constraints were set so that approximately 25%  
975 and 10% of architectures in the search space satisfy the constraints. The relative error compared to the optimal  
976 architecture satisfying the hardware constraints is reported between parentheses.  
977

Method	Top-1 accuracy (%)			Runs satisfied
	CIFAR-10	CIFAR-100	ImageNet16-120	
$c_{ss, param25} : \#param. \leq 187210$ (8193 architectures satisfied)				
<b>ConNAS</b>	<b>91.33 <math>\pm</math> 0.18 (-0.50)</b>	<b>64.03 <math>\pm</math> 0.66 (-2.55)</b>	<b>38.25 <math>\pm</math> 0.29 (-1.01)</b>	5 / 5
ProxylessNAS	85.87 $\pm$ 1.16 (-5.96)	51.11 $\pm$ 2.35 (-15.47)	25.27 $\pm$ 2.13 (-13.99)	5 / 5
FBNet	93.22 $\pm$ 0.20	69.43 $\pm$ 0.22	45.68 $\pm$ 0.37	0 / 5
TAS	91.55 $\pm$ 0.31	64.09 $\pm$ 0.61	39.01 $\pm$ 1.39	0 / 5
TF-NAS	89.43 $\pm$ 0.64 (-2.40)	58.75 $\pm$ 2.84 (-7.83)	31.99 $\pm$ 1.56 (-7.28)	5 / 5
HDX	79.79 $\pm$ 0.00 (-12.04)	32.30 $\pm$ 0.00 (-34.28)	15.53 $\pm$ 0.00 (-23.73)	5 / 5
$c_{ss, param10} : \#param. \leq 78906$ (3278 architectures satisfied)				
<b>ConNAS</b>	<b>90.04 <math>\pm</math> 0.53 (-1.06)</b>	<b>60.86 <math>\pm</math> 2.01 (-3.34)</b>	<b>34.09 <math>\pm</math> 0.83 (-2.71)</b>	5 / 5
ProxylessNAS	85.87 $\pm$ 1.16 (-5.23)	51.11 $\pm$ 2.35 (-13.09)	25.27 $\pm$ 2.13 (-11.53)	5 / 5
FBNet	93.22 $\pm$ 0.20	69.43 $\pm$ 0.22	45.68 $\pm$ 0.37	0 / 5
TAS	91.08 $\pm$ 0.40	63.75 $\pm$ 0.09	37.04 $\pm$ 0.83	0 / 5
TF-NAS	85.62 $\pm$ 0.00 (-5.48)	47.74 $\pm$ 0.00 (-16.46)	25.43 $\pm$ 0.00 (-11.37)	5 / 5
HDX	79.79 $\pm$ 0.00 (-11.31)	32.30 $\pm$ 0.00 (-31.90)	15.53 $\pm$ 0.00 (-21.27)	5 / 5
$c_{ss, memory25} : \text{peak mem.} \leq 393kB$ (12288 architectures satisfied)				
<b>ConNAS</b>	<b>92.91 <math>\pm</math> 0.08 (-0.21)</b>	69.28 $\pm$ 0.24 (-1.20)	43.65 $\pm$ 0.10 (-1.05)	5 / 5
ProxylessNAS	91.33 $\pm$ 0.19 (-1.79)	68.22 $\pm$ 0.28 (-2.26)	38.49 $\pm$ 0.03 (-6.21)	5 / 5
FBNet	93.16 $\pm$ 0.29	69.69 $\pm$ 0.05	45.53 $\pm$ 0.87	0 / 5
TAS	92.82 $\pm$ 0.28 (-0.30)	69.40 $\pm$ 0.46 (-1.08)	<b>43.69 <math>\pm</math> 0.49 (-1.01)</b>	5 / 5
TF-NAS	92.35 $\pm$ 0.09 (-0.77)	<b>69.52 <math>\pm</math> 0.13 (-0.96)</b>	43.09 $\pm$ 0.61 (-1.61)	5 / 5
HDX	79.79 $\pm$ 0.00 (-13.33)	32.30 $\pm$ 0.00 (-38.18)	15.53 $\pm$ 0.00 (-29.17)	5 / 5
$c_{ss, memory10} : \text{peak mem.} \leq 229kB$ (3584 architectures satisfied)				
<b>ConNAS</b>	<b>92.30 <math>\pm</math> 0.17 (-0.50)</b>	<b>68.80 <math>\pm</math> 0.32 (-0.50)</b>	<b>40.80 <math>\pm</math> 0.65 (-1.30)</b>	5 / 5
ProxylessNAS	91.33 $\pm$ 0.19 (-1.47)	68.22 $\pm$ 0.28 (-1.08)	38.49 $\pm$ 0.03 (-3.61)	5 / 5
FBNet	93.16 $\pm$ 0.29	69.69 $\pm$ 0.05	45.53 $\pm$ 0.87	0 / 5
TAS	92.31 $\pm$ 0.16	68.67 $\pm$ 0.46	41.98 $\pm$ 0.91	0 / 5
TF-NAS	91.33 $\pm$ 0.19 (-1.47)	68.22 $\pm$ 0.28 (-1.08)	38.49 $\pm$ 0.03 (-3.61)	5 / 5
HDX	79.79 $\pm$ 0.00 (-13.01)	32.30 $\pm$ 0.00 (-37.00)	15.53 $\pm$ 0.00 (-26.57)	5 / 5
$c_{ss, flops25} : \#FLOPs \leq 147235M$ (8193 architectures satisfied)				
<b>ConNAS</b>	91.58 $\pm$ 0.12 (-0.70)	64.94 $\pm$ 0.60 (-3.98)	<b>39.53 <math>\pm</math> 0.97 (-1.77)</b>	5 / 5
<b>ProxylessNAS</b>	<b>91.59 <math>\pm</math> 0.21 (-0.69)</b>	65.68 $\pm$ 0.85 (-3.24)	37.84 $\pm$ 0.62 (-3.46)	5 / 5
FBNet	92.87 $\pm$ 0.23	69.42 $\pm$ 0.60	44.23 $\pm$ 0.39	0 / 5
TAS	92.19 $\pm$ 0.13	68.06 $\pm$ 0.44	41.76 $\pm$ 0.04	0 / 5
TF-NAS	90.88 $\pm$ 0.02 (-1.40)	<b>65.87 <math>\pm</math> 0.56 (-3.05)</b>	38.09 $\pm$ 0.33 (-3.21)	5 / 5
HDX	79.79 $\pm$ 0.00 (-12.49)	32.30 $\pm$ 0.00 (-36.62)	15.53 $\pm$ 0.00 (-25.77)	5 / 5
$c_{ss, flops10} : \#FLOPs \leq 67063M$ (3278 architectures satisfied)				
<b>ConNAS</b>	<b>90.96 <math>\pm</math> 0.40 (-0.43)</b>	<b>63.48 <math>\pm</math> 0.53 (-3.10)</b>	<b>36.57 <math>\pm</math> 0.77 (-2.43)</b>	5 / 5
ProxylessNAS	91.37 $\pm$ 0.50	65.30 $\pm$ 1.07	37.63 $\pm$ 0.63	2 / 5
FBNet	92.87 $\pm$ 0.23	69.42 $\pm$ 0.60	44.23 $\pm$ 0.39	0 / 5
TAS	91.95 $\pm$ 0.04	66.54 $\pm$ 0.16	39.33 $\pm$ 0.91	0 / 5
TF-NAS	89.59 $\pm$ 0.60 (-1.80)	58.31 $\pm$ 1.52 (-8.27)	32.15 $\pm$ 0.69 (-6.85)	5 / 5
HDX	79.79 $\pm$ 0.00 (-11.60)	32.30 $\pm$ 0.00 (-34.28)	15.53 $\pm$ 0.00 (-23.47)	5 / 5

1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

1026

$c_{ss,comb25} : \#param. \leq 187210 \wedge \text{peak mem.} \leq 393kB \wedge$ $\#FLOPs \leq 147235M$ (5364 architectures satisfied)				
<b>ConNAS</b>	<b>91.29 <math>\pm</math> 0.11 (-0.37)</b>	<b>64.88 <math>\pm</math> 0.54 (-1.70)</b>	<b>38.03 <math>\pm</math> 0.09 (-1.24)</b>	5 / 5
ProxylessNAS	87.76 $\pm$ 0.59 (-3.90)	56.22 $\pm$ 1.18 (-10.36)	26.39 $\pm$ 0.37 (-12.87)	5 / 5
FBNet	92.88 $\pm$ 0.19	68.83 $\pm$ 0.34	43.88 $\pm$ 0.42	0 / 5
TAS	91.90 $\pm$ 0.11	66.93 $\pm$ 0.13	40.39 $\pm$ 0.26	0 / 5
TF-NAS	90.59 $\pm$ 0.19 (-1.07)	63.30 $\pm$ 0.59 (-3.28)	36.58 $\pm$ 0.55 (-2.69)	5 / 5
HDX	79.79 $\pm$ 0.00 (-11.87)	32.30 $\pm$ 0.00 (-34.28)	15.53 $\pm$ 0.00 (-23.73)	5 / 5

$c_{ss,comb10} : \#param. \leq 78906 \wedge \text{peak mem.} \leq 229kB \wedge$ $\#FLOPs \leq 67063M$ 970 architectures satisfied				
<b>ConNAS</b>	<b>89.43 <math>\pm</math> 0.20 (-1.00)</b>	<b>61.70 <math>\pm</math> 0.46 (-1.38)</b>	<b>31.78 <math>\pm</math> 1.01 (-2.52)</b>	5 / 5
ProxylessNAS	87.76 $\pm$ 0.59 (-2.67)	56.22 $\pm$ 1.18 (-6.86)	26.39 $\pm$ 0.37 (-7.91)	5 / 5
FBNet	92.88 $\pm$ 0.19	68.83 $\pm$ 0.34	43.88 $\pm$ 0.42	0 / 5
TAS	91.10 $\pm$ 0.19	64.45 $\pm$ 0.02	36.96 $\pm$ 0.06	0 / 5
TF-NAS	88.71 $\pm$ 0.13 (-1.72)	58.14 $\pm$ 1.13 (-4.94)	28.84 $\pm$ 1.24 (-5.46)	5 / 5
HDX	79.79 $\pm$ 0.00 (-10.64)	32.30 $\pm$ 0.00 (-30.78)	15.53 $\pm$ 0.00 (-18.77)	5 / 5

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080 **D ABLATION WEIGHTING FACTOR BASELINE METHODS**  
10811082 **Table 9: Comparison of classification results on NATS-Bench topology search space across baseline**  
1083 **methods and weighting factors.** Test accuracy (mean  $\pm$  std over 5 runs) is reported under hardware constraints,  
1084 using on-device measurements of inference latency and energy usage on an NVIDIA Jetson TX2 Edge GPU.  
1085 The relative error compared to the optimal architecture satisfying the hardware constraints is reported between  
1086 parentheses. Best results for each method, constraint, and dataset are highlighted in bold. Performance of  
1087 experiments that do not satisfy the hardware constraints are reported in gray.  
1088

1089	Method	$\lambda_{\text{hardware}}$	Top-1 accuracy (%)			Runs satisfied
			CIFAR-10	CIFAR-100	ImageNet16-120	
<i><math>c_{\text{ts},\text{latency}} : \text{latency} \leq 5.31ms</math> (7814 architectures satisfied)</i>						
1091	ProxylessNAS	0.5	92.06 $\pm$ 0.13 (-2.20)	67.15 $\pm$ 0.18 (-6.01)	39.23 $\pm$ 0.09 (-7.37)	5 / 5
1092	<b>ProxylessNAS</b>	<b>1.0</b>	<b>92.29 <math>\pm</math> 0.00 (-1.97)</b>	<b>67.48 <math>\pm</math> 0.00 (-5.68)</b>	<b>39.40 <math>\pm</math> 0.00 (-7.20)</b>	5 / 5
1093	ProxylessNAS	1.5	42.92 $\pm$ 45.07 (-51.34)	27.59 $\pm$ 36.41 (-45.57)	16.26 $\pm$ 21.12 (-30.34)	5 / 5
1094	FBNet	0.5	93.10 $\pm$ 0.09 (-1.16)	69.94 $\pm$ 0.07 (-3.22)	39.67 $\pm$ 0.05 (-6.93)	5 / 5
1095	FBNet	0.6	93.18 $\pm$ 0.12 (-1.08)	70.00 $\pm$ 0.08 (-3.16)	39.71 $\pm$ 0.06 (-6.89)	5 / 5
1096	<b>FBNet</b>	<b>0.7</b>	<b>93.27 <math>\pm</math> 0.00 (-1.00)</b>	<b>70.06 <math>\pm</math> 0.00 (-3.10)</b>	<b>39.75 <math>\pm</math> 0.00 (-6.85)</b>	5 / 5
1097	TAS	0.5	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
1098	TAS	1.0	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
1099	TAS	1.5	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
1100	<b>TF-NAS</b>	<b>0.5</b>	<b>93.27 <math>\pm</math> 0.00 (-1.00)</b>	<b>70.06 <math>\pm</math> 0.00 (-3.10)</b>	<b>39.75 <math>\pm</math> 0.00 (-6.85)</b>	5 / 5
1101	<b>TF-NAS</b>	<b>1.0</b>	<b>93.27 <math>\pm</math> 0.00 (-1.00)</b>	<b>70.06 <math>\pm</math> 0.00 (-3.10)</b>	<b>39.75 <math>\pm</math> 0.00 (-6.85)</b>	5 / 5
1102	TF-NAS	1.5	93.01 $\pm$ 0.57 (-1.25)	69.48 $\pm$ 1.29 (-3.68)	39.62 $\pm$ 0.30 (-6.98)	5 / 5
1103	HDX	0.5	42.86 $\pm$ 44.99 (-51.40)	27.51 $\pm$ 36.30 (-45.65)	16.22 $\pm$ 21.07 (-30.38)	5 / 5
1104	<b>HDX</b>	<b>1.0</b>	<b>75.83 <math>\pm</math> 36.80 (-18.43)</b>	<b>54.18 <math>\pm</math> 29.73 (-18.98)</b>	<b>31.69 <math>\pm</math> 17.25 (-14.91)</b>	5 / 5
1105	HDX	1.5	59.37 $\pm$ 45.07 (-34.89)	40.89 $\pm$ 36.41 (-32.27)	23.97 $\pm$ 21.12 (-22.63)	5 / 5
<i><math>c_{\text{ts},\text{energy}} : \text{energy usage} \leq 23.95mJ</math> (7814 architectures satisfied)</i>						
1106	ProxylessNAS	0.5	<b>92.29 <math>\pm</math> 0.00 (-1.86)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.46)</b>	<b>39.40 <math>\pm</math> 0.00 (-7.37)</b>	5 / 5
1107	<b>ProxylessNAS</b>	<b>1.0</b>	<b>92.29 <math>\pm</math> 0.00 (-1.86)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.46)</b>	<b>39.40 <math>\pm</math> 0.00 (-7.37)</b>	5 / 5
1108	ProxylessNAS	1.5	26.46 $\pm$ 36.80 (-67.69)	14.30 $\pm$ 29.73 (-57.64)	8.55 $\pm$ 17.25 (-38.22)	5 / 5
1109	FBNet	0.5	93.05 $\pm$ 0.00 (-1.10)	69.91 $\pm$ 0.00 (-2.03)	39.64 $\pm$ 0.00 (-7.12)	5 / 5
1110	FBNet	0.6	93.05 $\pm$ 0.00 (-1.10)	69.91 $\pm$ 0.00 (-2.03)	39.64 $\pm$ 0.00 (-7.12)	5 / 5
1111	<b>FBNet</b>	<b>0.7</b>	<b>93.22 <math>\pm</math> 0.09 (-0.93)</b>	<b>70.03 <math>\pm</math> 0.07 (-1.91)</b>	<b>39.73 <math>\pm</math> 0.05 (-7.04)</b>	5 / 5
1112	TAS	0.5	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
1113	TAS	1.0	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
1114	TAS	1.5	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
1115	<b>TF-NAS</b>	<b>0.5</b>	<b>93.27 <math>\pm</math> 0.00 (-0.89)</b>	<b>70.06 <math>\pm</math> 0.00 (-1.88)</b>	<b>39.75 <math>\pm</math> 0.00 (-7.02)</b>	5 / 5
1116	TF-NAS	1.0	93.01 $\pm$ 0.57 (-1.14)	69.48 $\pm$ 1.29 (-2.46)	39.62 $\pm$ 0.30 (-7.15)	5 / 5
1117	TF-NAS	1.5	91.99 $\pm$ 0.01 (-2.16)	67.15 $\pm$ 0.04 (-4.79)	39.11 $\pm$ 0.04 (-7.66)	5 / 5
1118	HDX	0.5	42.92 $\pm$ 45.07 (-51.23)	27.59 $\pm$ 36.41 (-44.35)	16.26 $\pm$ 21.12 (-30.51)	5 / 5
1119	HDX	1.0	92.29 $\pm$ 0.00 (-1.86)	67.48 $\pm$ 0.00 (-4.46)	39.40 $\pm$ 0.00 (-7.13)	5 / 5
1120	HDX	1.5	59.37 $\pm$ 45.07 (-34.89)	40.89 $\pm$ 36.41 (-32.27)	23.97 $\pm$ 21.12 (-22.63)	5 / 5
<i><math>c_{\text{ts},\text{comb}} : \text{latency} \leq 5.31ms \wedge \text{energy usage} \leq 23.95mJ</math> (7586 architectures satisfied)</i>						
1121	ProxylessNAS	0.5	<b>92.29 <math>\pm</math> 0.00 (-1.86)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.46)</b>	<b>39.40 <math>\pm</math> 0.00 (-7.13)</b>	5 / 5
1122	<b>ProxylessNAS</b>	<b>1.0</b>	<b>92.29 <math>\pm</math> 0.00 (-1.86)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.46)</b>	<b>39.40 <math>\pm</math> 0.00 (-7.13)</b>	5 / 5
1123	ProxylessNAS	1.5	42.92 $\pm$ 45.07 (-51.23)	27.59 $\pm$ 36.41 (-44.35)	16.26 $\pm$ 21.12 (-30.27)	5 / 5
1124	<b>FBNet</b>	<b>0.7</b>	<b>92.01 <math>\pm</math> 0.00 (-2.14)</b>	<b>67.07 <math>\pm</math> 0.00 (-4.87)</b>	<b>39.19 <math>\pm</math> 0.00 (-7.34)</b>	5 / 5
1125	<b>FBNet</b>	<b>0.6</b>	<b>92.01 <math>\pm</math> 0.00 (-2.14)</b>	<b>67.07 <math>\pm</math> 0.00 (-4.87)</b>	<b>39.19 <math>\pm</math> 0.00 (-7.34)</b>	5 / 5
1126	<b>FBNet</b>	<b>0.7</b>	<b>92.01 <math>\pm</math> 0.00 (-2.14)</b>	<b>67.07 <math>\pm</math> 0.00 (-4.87)</b>	<b>39.19 <math>\pm</math> 0.00 (-7.34)</b>	5 / 5
1127	TAS	0.5	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
1128	TAS	1.0	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
1129	TAS	1.5	93.67 $\pm$ 0.00	70.91 $\pm$ 0.00	41.02 $\pm$ 0.00	0 / 5
1130	<b>TF-NAS</b>	<b>0.5</b>	<b>93.27 <math>\pm</math> 0.00 (-0.89)</b>	<b>70.06 <math>\pm</math> 0.00 (-1.88)</b>	<b>39.75 <math>\pm</math> 0.00 (-6.78)</b>	5 / 5
1131	TF-NAS	<b>1.0</b>	<b>93.27 <math>\pm</math> 0.00 (-0.89)</b>	<b>70.06 <math>\pm</math> 0.00 (-1.88)</b>	<b>39.75 <math>\pm</math> 0.00 (-6.78)</b>	5 / 5
1132	TF-NAS	1.5	92.76 $\pm$ 0.69 (-1.39)	68.89 $\pm$ 1.61 (-3.05)	39.51 $\pm$ 0.34 (-7.02)	5 / 5
1133	HDX	0.5	92.23 $\pm$ 0.13	67.40 $\pm$ 0.18	39.36 $\pm$ 0.09	1 / 5
1134	<b>HDX</b>	<b>1.0</b>	<b>92.29 <math>\pm</math> 0.00 (-1.86)</b>	<b>67.48 <math>\pm</math> 0.00 (-4.46)</b>	<b>39.40 <math>\pm</math> 0.00 (-7.13)</b>	5 / 5
1135	HDX	1.5	59.37 $\pm$ 45.07 (-34.78)	40.89 $\pm$ 36.41 (-31.05)	23.97 $\pm$ 21.12 (-22.56)	5 / 5

1134  
 1135 **Table 10: Comparison of classification results on NATS-Bench size search space across baseline meth-**  
 1136 **ods and weighting factors.** Test accuracy (mean  $\pm$  std over 5 runs) is reported under hardware constraints,  
 1137 including the number of parameters, number of FLOPs, and peak memory usage. The relative error compared  
 1138 to the optimal architecture satisfying the hardware constraints is reported between parentheses. Best results for  
 1139 each method, constraint, and dataset are highlighted in bold. Performance of experiments that do not satisfy  
 the hardware constraints are reported in gray.

1140	Method	$\lambda_{\text{hardware}}$	Top-1 accuracy (%)			Runs satisfied
			CIFAR-10	CIFAR-100	ImageNet16-120	
<i>C<sub>ss,param</sub> : #parameters <math>\leq 261650</math> (16385 architectures satisfied)</i>						
1143	<b>ProxylessNAS</b>	<b>0.5</b>	<b>90.87 <math>\pm</math> 0.02 (-1.66)</b>	<b>61.31 <math>\pm</math> 0.61 (-7.61)</b>	<b>35.34 <math>\pm</math> 0.31 (-6.69)</b>	5 / 5
1144	ProxylessNAS	1.0	85.87 $\pm$ 1.16 (-6.66)	51.11 $\pm$ 2.35 (-17.81)	25.27 $\pm$ 2.13 (-16.76)	5 / 5
1145	ProxylessNAS	1.5	83.47 $\pm$ 0.74 (-9.06)	45.36 $\pm$ 2.95 (-23.56)	21.59 $\pm$ 1.01 (-20.45)	5 / 5
1146	FBNet	0.5	93.12 $\pm$ 0.17	69.36 $\pm$ 0.44	45.43 $\pm$ 0.48	0 / 5
1147	FBNet	0.6	93.22 $\pm$ 0.20	69.43 $\pm$ 0.22	45.68 $\pm$ 0.37	0 / 5
1148	FBNet	0.7	92.91 $\pm$ 0.26	69.36 $\pm$ 0.24	45.40 $\pm$ 0.41	0 / 5
1149	TAS	0.5	92.26 $\pm$ 0.01	67.41 $\pm$ 0.38	42.10 $\pm$ 0.15	0 / 5
1150	TAS	1.0	92.25 $\pm$ 0.02	67.34 $\pm$ 0.54	41.89 $\pm$ 0.33	0 / 5
1151	TAS	1.5	92.25 $\pm$ 0.02	67.34 $\pm$ 0.54	41.89 $\pm$ 0.33	0 / 5
1152	TF-NAS	0.5	91.30 $\pm$ 0.19 (-1.23)	<b>64.88 <math>\pm</math> 0.28 (-4.04)</b>	<b>40.43 <math>\pm</math> 0.24 (-1.61)</b>	5 / 5
1153	<b>TF-NAS</b>	<b>1.0</b>	<b>91.78 <math>\pm</math> 0.00 (-0.75)</b>	63.64 $\pm$ 0.00 (-5.28)	38.00 $\pm$ 0.00 (-4.03)	5 / 5
1154	TF-NAS	1.5	91.56 $\pm$ 0.30 (-0.97)	64.05 $\pm$ 0.56 (-4.87)	38.17 $\pm$ 0.24 (-3.86)	5 / 5
1155	<b>HDX</b>	<b>0.5</b>	<b>88.00 <math>\pm</math> 0.91 (-4.53)</b>	<b>54.74 <math>\pm</math> 1.31 (-14.18)</b>	<b>29.33 <math>\pm</math> 3.50 (-12.71)</b>	5 / 5
1156	HDX	1.0	86.10 $\pm$ 1.28 (-6.43)	51.71 $\pm$ 2.88 (-17.21)	25.36 $\pm$ 2.42 (-16.67)	5 / 5
1157	HDX	1.5	83.37 $\pm$ 0.88 (-9.16)	45.20 $\pm$ 3.12 (-23.72)	21.54 $\pm$ 1.51 (-20.49)	5 / 5
<i>C<sub>ss,mem</sub> : peak memory usage <math>\leq 655kB</math> (20480 architectures satisfied)</i>						
1158	ProxylessNAS	0.5	<b>91.61 <math>\pm</math> 0.09 (-1.81)</b>	67.90 $\pm$ 0.24 (-2.96)	38.30 $\pm$ 0.18 (-7.63)	5 / 5
1159	ProxylessNAS	1.0	91.33 $\pm$ 0.19 (-2.09)	<b>68.22 <math>\pm</math> 0.28 (-2.64)</b>	<b>38.49 <math>\pm</math> 0.03 (-7.45)</b>	5 / 5
1160	ProxylessNAS	1.5	91.33 $\pm$ 0.19 (-2.09)	<b>68.22 <math>\pm</math> 0.28 (-2.64)</b>	<b>38.49 <math>\pm</math> 0.03 (-7.45)</b>	5 / 5
1161	FBNet	0.5	93.24 $\pm$ 0.17	70.03 $\pm$ 0.38	45.43 $\pm$ 0.55	0 / 5
1162	FBNet	0.6	93.16 $\pm$ 0.29	69.69 $\pm$ 0.05	45.53 $\pm$ 0.87	1 / 5
1163	FBNet	0.7	93.23 $\pm$ 0.16	69.94 $\pm$ 0.34	45.71 $\pm$ 0.45	1 / 5
1164	TAS	0.5	93.08 $\pm$ 0.20 (-0.34)	70.23 $\pm$ 0.16 (-0.63)	<b>45.14 <math>\pm</math> 0.31 (-0.79)</b>	5 / 5
1165	TAS	1.0	92.93 $\pm$ 0.19 (-0.49)	70.10 $\pm$ 0.32 (-0.76)	44.98 $\pm$ 0.59 (-0.95)	5 / 5
1166	TAS	1.5	<b>93.13 <math>\pm</math> 0.19 (-0.29)</b>	<b>70.30 <math>\pm</math> 0.19 (-0.56)</b>	45.02 $\pm$ 0.32 (-0.91)	5 / 5
1167	TF-NAS	0.5	92.92 $\pm$ 0.31 (-0.50)	69.94 $\pm$ 0.31 (-0.92)	44.73 $\pm$ 0.67 (-1.21)	5 / 5
1168	<b>TF-NAS</b>	<b>1.0</b>	<b>92.99 <math>\pm</math> 0.10 (-0.43)</b>	<b>70.21 <math>\pm</math> 0.16 (-0.65)</b>	<b>45.25 <math>\pm</math> 0.27 (-0.69)</b>	5 / 5
1169	TF-NAS	1.5	<b>93.02 <math>\pm</math> 0.28 (-0.40)</b>	70.13 $\pm$ 0.34 (-0.73)	45.02 $\pm$ 0.61 (-0.91)	5 / 5
1170	HDX	0.5	89.43 $\pm$ 5.40 (-3.99)	60.85 $\pm$ 15.96 (-10.01)	34.23 $\pm$ 10.49 (-11.70)	5 / 5
1171	HDX	1.0	<b>91.59 <math>\pm</math> 0.19 (-1.83)</b>	68.03 $\pm$ 0.29 (-2.83)	38.41 $\pm$ 0.07 (-7.53)	5 / 5
1172	HDX	1.5	91.41 $\pm$ 0.24 (-2.01)	<b>68.09 <math>\pm</math> 0.34 (-2.77)</b>	<b>38.47 <math>\pm</math> 0.04 (-7.46)</b>	5 / 5
<i>C<sub>ss,flops</sub> : #FLOPs <math>\leq 344194M</math> (16385 architectures satisfied)</i>						
1173	<b>ProxylessNAS</b>	<b>0.5</b>	<b>91.98 <math>\pm</math> 0.00 (-0.99)</b>	<b>66.66 <math>\pm</math> 0.00 (-3.82)</b>	<b>40.00 <math>\pm</math> 0.00 (-4.37)</b>	5 / 5
1174	ProxylessNAS	1.0	91.59 $\pm$ 0.21 (-1.38)	65.68 $\pm$ 0.85 (-4.80)	37.84 $\pm$ 0.62 (-6.53)	5 / 5
1175	ProxylessNAS	1.5	90.82 $\pm$ 0.12 (-2.15)	63.58 $\pm$ 0.91 (-6.90)	36.03 $\pm$ 0.59 (-8.33)	5 / 5
1176	FBNet	0.5	92.92 $\pm$ 0.14	69.71 $\pm$ 0.52	44.52 $\pm$ 0.76	0 / 5
1177	FBNet	0.6	92.87 $\pm$ 0.23	69.42 $\pm$ 0.60	44.23 $\pm$ 0.39	0 / 5
1178	FBNet	0.7	92.92 $\pm$ 0.17	69.42 $\pm$ 0.59	44.20 $\pm$ 0.58	0 / 5
1179	TAS	0.5	92.70 $\pm$ 0.18	69.10 $\pm$ 0.24	44.08 $\pm$ 0.18	0 / 5
1180	TAS	1.0	92.57 $\pm$ 0.23	68.36 $\pm$ 0.68	42.71 $\pm$ 0.46	0 / 5
1181	TAS	1.5	92.33 $\pm$ 0.00	67.66 $\pm$ 0.00	43.27 $\pm$ 0.00	0 / 5
1182	<b>TF-NAS</b>	<b>0.5</b>	<b>92.22 <math>\pm</math> 0.07 (-0.75)</b>	<b>68.32 <math>\pm</math> 0.55 (-2.16)</b>	<b>41.93 <math>\pm</math> 0.40 (-2.44)</b>	5 / 5
1183	TF-NAS	1.0	92.19 $\pm$ 0.18 (-0.78)	67.81 $\pm$ 0.56 (-2.67)	41.69 $\pm$ 1.16 (-2.68)	5 / 5
1184	TF-NAS	1.5	<b>92.22 <math>\pm</math> 0.18 (-0.75)</b>	67.61 $\pm$ 0.24 (-2.87)	41.83 $\pm$ 1.19 (-2.53)	5 / 5
1185	HDX	0.5	81.86 $\pm$ 2.83 (-11.11)	40.12 $\pm$ 10.70 (-30.36)	18.75 $\pm$ 4.40 (-25.62)	5 / 5
1186	<b>HDX</b>	<b>1.0</b>	<b>86.88 <math>\pm</math> 0.00 (-6.09)</b>	<b>55.42 <math>\pm</math> 0.00 (-15.06)</b>	<b>28.70 <math>\pm</math> 0.00 (-15.67)</b>	5 / 5
1187	HDX	1.5	86.25 $\pm$ 0.00 (-6.72)	54.56 $\pm$ 0.00 (-15.92)	26.80 $\pm$ 0.00 (-17.57)	5 / 5

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

$c_{ss,comb} : \#parameters \leq 261650 \wedge \text{peak mem.} \leq 655kB \wedge$ $\#FLOPs \leq 344194M$ (13342 architectures satisfied)					
<b>ProxylessNAS</b>	<b>0.5</b>	<b>90.66 ± 0.11 (-1.87)</b>	<b>63.04 ± 0.34 (-5.88)</b>	<b>35.59 ± 0.19 (-6.44)</b>	5 / 5
ProxylessNAS	1.0	87.76 ± 0.59 (-4.77)	56.22 ± 1.18 (-12.70)	26.39 ± 0.37 (-15.64)	5 / 5
ProxylessNAS	1.5	83.99 ± 0.00 (-8.54)	48.64 ± 0.00 (-20.28)	22.50 ± 0.00 (-19.53)	5 / 5
FBNet	0.5	92.87 ± 0.12	68.90 ± 0.31	44.00 ± 0.09	0 / 5
FBNet	0.6	92.88 ± 0.19	68.83 ± 0.34	43.88 ± 0.42	0 / 5
FBNet	0.7	92.63 ± 0.19	68.64 ± 0.74	42.95 ± 1.10	0 / 5
TAS	0.5	92.53 ± 0.14	67.69 ± 0.63	42.78 ± 0.57	0 / 5
TAS	1.0	92.26 ± 0.01	67.47 ± 0.42	41.97 ± 0.35	0 / 5
TAS	1.5	92.26 ± 0.00	67.58 ± 0.00	42.03 ± 0.00	0 / 5
TF-NAS	0.5	91.84 ± 0.07 (-0.69)	<b>66.89 ± 0.39 (-2.03)</b>	39.31 ± 0.53 (-2.72)	5 / 5
TF-NAS	1.0	91.76 ± 0.00 (-0.77)	<b>66.46 ± 0.00 (-2.46)</b>	38.73 ± 0.00 (-3.30)	5 / 5
<b>TF-NAS</b>	<b>1.5</b>	<b>91.85 ± 0.21 (-0.68)</b>	<b>66.64 ± 0.16 (-2.28)</b>	<b>40.07 ± 0.91 (-1.97)</b>	5 / 5
HDX	0.5	79.79 ± 0.00 (-12.74)	32.30 ± 0.00 (-36.62)	15.53 ± 0.00 (-26.50)	5 / 5
<b>HDX</b>	<b>1.0</b>	<b>84.96 ± 0.00 (-7.57)</b>	<b>51.84 ± 0.00 (-17.08)</b>	<b>23.57 ± 0.00 (-18.47)</b>	5 / 5
HDX	1.5	83.93 ± 2.31 (-8.60)	47.93 ± 8.74 (-20.99)	21.96 ± 3.59 (-20.07)	5 / 5

1242 **E TRAINING DETAILS**  
12431244 To facilitate reproducibility, we provide additional experimental details in this appendix.<sup>2</sup>  
12451246 **E.1 NATS-BENCH**  
12471248 We adopt the same data augmentation techniques and training procedure as described in Dong et al.  
1249 (2021). The search process is performed in two stages.  
12501251 In the first stage, a supernet is trained on 50% of the CIFAR-10 training set (25,000 images) for  
1252 150 epochs, using a batch size of 64. The operation weights are optimized via Nesterov momentum  
1253 Stochastic Gradient Descent (SGD) with a momentum of 0.9 and a weight decay of  $5 \times 10^{-4}$ . The  
1254 initial learning rate is set to 0.0025 and annealed to 0.001 over 100 epochs using a cosine schedule.  
1255 Architecture parameters are optimized using Adam with a learning rate of 0.001, weight decay of  
1256  $1 \times 10^{-3}$ , and  $\beta_1 = 0.5$ ,  $\beta_2 = 0.999$ . The sampling temperature starts at 10 and is linearly annealed  
1257 to 0.1 over 100 epochs. The rescale factor  $R$  is kept constant throughout training (we use  $R = 1.2$   
1258 for most experiments; an ablation study on the rescale factor is presented in Appendix B).  
12591260 In the second stage, an architecture is selected based on the lowest cross-entropy loss, evaluated  
1261 on the remaining 50% of the training set, while satisfying the imposed hardware constraints. The  
1262 NATS-Bench performance lookup table is then used to retrieve the test accuracy of the sampled  
1263 architecture on CIFAR-10, CIFAR-100, and ImageNet16-120.  
12641265 All experiments are repeated five times using fixed random seeds. The search is performed on a  
1266 single NVIDIA V100 GPU and took approximately 5 hours to complete.  
12671268 **E.2 CONDITION MONITORING USE CASE**  
12691270 For the condition monitoring use case, a dataset is created from Desenfans et al. (2025), representing  
1271 readings from a voltage meter, a current sensor and an encoder.<sup>3</sup> The complete dataset contains  
1272 31,920 time series instances, split into 25,530 train for training and 6,390 for testing. Each instance  
1273 consists of 256 time steps with 8 features: 3-phase current, 3-phase voltage, motor speed, and rotor  
1274 angle. The instances are labeled as either *no fault*, *static eccentricity fault*, *dynamic eccentricity*  
1275 *fault*, or *mixed eccentricity fault*. A Fast Fourier Transform (FFT) is applied to the 3-phase current  
1276 and voltage signals, and all 8 features are normalized to the range [0, 1]. The search process is again  
1277 performed in two stages.  
12781279 In the first stage, a supernet is trained on 80% of the training set (20424 instances) for 150 epochs,  
1280 using a batch size of 128. The operation weights are optimized via Nesterov momentum Stochastic  
1281 Gradient Descent (SGD) with a momentum of 0.9 and a weight decay of  $5 \times 10^{-4}$ . The initial learning  
1282 rate is set to 0.0025 and annealed to 0.001 over 100 epochs using a cosine schedule. Architecture  
1283 parameters are optimized using Adam with a learning rate of 0.001, weight decay of  $1 \times 10^{-3}$ , and  
1284  $\beta_1 = 0.5$ ,  $\beta_2 = 0.999$ . The sampling temperature starts at 10 and is linearly annealed to 0.1 over  
1285 100 epochs. The rescale factor  $R$  is kept constant at 1.2.  
12861287 In the second stage, an architecture is selected based on the lowest cross-entropy loss, evaluated  
1288 on the remaining 20% of the training set, while satisfying the imposed hardware constraints. The  
1289 selected architectures are trained using 5-fold cross-validation, where each fold uses 80% of the  
1290 training data for training over 250 epochs with a batch size of 16. Optimization is performed using  
1291 SGD with a momentum of 0.9 and a weight decay of  $1 \times 10^{-4}$ . The initial learning rate is set to  
1292 0.0025 and annealed to 0.0001 using a cosine scheduler.  
12931294 All experiments are repeated five times using fixed random seeds. The search is performed on a  
1295 single NVIDIA P100 GPU and took approximately 7 hours to complete.  
12961297 <sup>2</sup>The code required to run all experiments will be made publicly available upon publication of the paper.  
12981299 <sup>3</sup>The dataset will be made publicly available upon publication of the paper.  
1300

1296 F CONDITION MONITORING SEARCH SPACE  
12971298 The search space for the condition monitoring use case is defined as a supernet composed of eight  
1299 consecutive edges. Each edge can select from the following candidate operations:  
13001301 • A 1D convolution followed by a batch normalization and a ReLU activation  
1302 • A 1D depthwise separable convolution, where each convolution is followed by a batch  
1303 normalization and a ReLU activation  
1304 • An identity operation (available only on even-numbered edges)  
13051306 The classification head consists of a global average pooling layer followed by a fully connected  
1307 layer.  
13081309 For each convolution, the number of output channels can be chosen from  $\{16, 32, 64, 256, 512\}$ .  
1310 Additionally, a stride of 2 is applied to convolutions on odd-numbered edges. Convolutional filters  
1311 are shared across operations, as proposed in Wan et al. (2020), to reduce the number of trainable  
1312 parameters and improve memory and computational efficiency.  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349

1350 **G CONDITION MONITORING HAND-CRAFTED BASELINES**  
13511352 To ensure coverage of diverse hardware constraints, we sampled six hand-crafted architectures from  
1353 the proposed search space. These include the smallest (Conv1D-x-Min) and largest (Conv1D-x-  
1354 Max) models in terms of latency and energy consumption, for both regular and depthwise-separable  
1355 convolutions (Chollet, 2017). Additionally, Conv1D-Reg and Conv1D-DS follow common design  
1356 principles where channel width increases as spatial dimensions decrease.  
13571358 **Table 11: Overview of hand-crafted architectures used for comparison.** Each convolutional block consists  
1359 of a convolutional operation regular or depthwise separable, a batch normalization, and a ReLU activation.

Name	Number of blocks	Kernel	Channels	Strides	Convolution type
Conv1D-Reg	4	3	[16, 32, 64, 128]	[2, 2, 2, 2]	Regular
Conv1D-DS	4	3	[16, 32, 64, 128]	[2, 2, 2, 2]	Depthwise Separable
Conv1D-Reg-Max	8	3	[512, 512, 512, 512, 512, 512, 512, 512]	[2, 1, 2, 1, 2, 1, 2, 1]	Regular
Conv1D-DS-Max	8	3	[512, 512, 512, 512, 512, 512, 512, 512]	[2, 1, 2, 1, 2, 1, 2, 1]	Depthwise Separable
Conv1D-Reg-Min	4	3	[8, 8, 8, 8]	[2, 2, 2, 2]	Regular
Conv1D-DS-Min	4	3	[8, 8, 8, 8]	[2, 2, 2, 2]	Depthwise Separable

1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403

1404 **H DISCLOSURE ON THE USE OF GENERATIVE ARTIFICIAL INTELLIGENCE**  
14051406 We used a large language model (LLM) as a writing aid. Ideation, scientific content, experimental  
1407 design, and analysis were conducted entirely by the authors without any contribution of an LLM.  
1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1445

1446

1447

1448

1449

1450

1451

1452

1453

1454

1455

1456

1457