Multi-Objective Hardware-Aware Neural Architecture Search with Pareto Rank-Preserving Surrogate Models

HADJER BENMEZIANE, Univ. Polytechnique Hauts-de-France, UMR 8201 - LAMIH - F-59313 Valenciennes, France
HAMZA OUARNOUGHI, Univ. Polytechnique Hauts-de-France, UMR 8201 - LAMIH - F-59313 Valenciennes, France
KAOUTAR EL MAGHRAOUI, IBM T. J. Watson Research Center Yorktown Heights, NY, USA
SMAIL NIAR, Univ. Polytechnique Hauts-de-France, UMR 8201 - LAMIH - F-59313 Valenciennes, France

Deep learning (DL) models such as convolutional neural networks (ConvNets) are being deployed to solve various computer vision and natural language processing tasks at the edge. It is a challenge to find the right DL architecture that simultaneously meets the accuracy, power and performance budgets of such resource-constrained devices. Hardware-aware Neural Architecture Search (HW-NAS) has recently gained steam by automating the design of efficient DL models for a variety of target hardware platform. However, such algorithms require excessive computational resources. Thousands of GPU days are required to evaluate and explore an architecture search space such as FBNet [45]. State-of-the-art approaches propose using surrogate models to predict architecture accuracy and hardware performance to speed up HW-NAS. Existing approaches use independent surrogate models to estimate each objective, resulting in non-optimal Pareto fronts. In this paper, HW-PR-NAS [1], a novel Pareto rank-preserving surrogate model for edge computing platforms is presented. Our model integrates a new loss function that ranks the architectures according to their Pareto rank, regardless of the actual values of the various objectives. We employ a simple yet effective surrogate model architecture that can be generalized to any standard DL model. We then present an optimized evolutionary algorithm that uses and validates our surrogate model. Our approach has been evaluated on seven edge hardware platforms from various classes, including ASIC, FPGA, GPU and multi-core CPU. The evaluation results show that HW-PR-NAS achieves up to 2.5x speedup compared to state-of-the-art methods while achieving 98% near the actual Pareto front.

CCS Concepts: • **Computing methodologies** → *Simulation evaluation*; *Object recognition*; Speech recognition; **Discrete space search**.

Additional Key Words and Phrases: Deep Neural Networks, Hardware-aware Neural Architecture, Design Space Exploration, Surrogate Models, Model Accuracy, Execution Time, Power Consumption

[1]Extension of conference paper: HW-PR-NAS [3]. In the conference paper, we proposed a Pareto rank-preserving surrogate model trained with a dedicated loss function. This article extends the conference paper by presenting a novel lightweight architecture for the surrogate model that enables faster inference and, thus, more efficient NAS. We also evaluate our HW-PR-NAS on an NLP use case, namely KWS, and validate that HW-PR-NAS only needs five epochs of fine-tuning to generalize to a new dataset and a new hardware platform.

Authors' addresses: Hadjer Benmeziane, hadjer.benmeziane@uphf.fr, Univ. Polytechnique Hauts-de-France, UMR 8201 - LAMIH - F-59313 Valenciennes, France; Hamza Ouarnoughi, hamza.ouarnoughi@uphf.fr, Univ. Polytechnique Hauts-de-France, UMR 8201 - LAMIH - F-59313 Valenciennes, France; Kaoutar El Maghraoui, kelmaghr@us.ibm.com, IBM T. J. Watson Research Center Yorktown Heights, NY, USA; Smail Niar, smail.niar@uphf.fr, Univ. Polytechnique Hauts-de-France, UMR 8201 - LAMIH - F-59313 Valenciennes, France.

1544-3566/2022/09-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smail Niar. 2022. Multi-Objective Hardware-Aware Neural Architecture Search with Pareto Rank-Preserving Surrogate Models. *ACM Trans. Arch. Code Optim.* 37, 4, Article 111 (September 2022), 22 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

With the rise of Automated Machine Learning (AutoML) techniques, significant progress has been made to automate ML and democratize Artificial Intelligence (AI) for the masses. Neural Architecture Search (NAS), a subset of AutoML, is a powerful technique that automates neural network design and frees Deep Learning (DL) researchers from the tedious and time-consuming task of handcrafting DL architectures [2]. Recently, NAS methods have exhibited remarkable advances reducing computational costs, improving accuracy, and even surpassing human performance on DL architecture design in several use cases such as image classification [12, 23] and object detection [24, 40].

As we are witnessing a massive increase in hardware diversity ranging from tiny Microcontroller Units (MCU) to server-class supercomputers, it has become crucial to design efficient neural networks adapted to various platforms. Traditional NAS techniques focus on searching for the most accurate architectures, overlooking the target hardware efficiency's practical aspects. Considering hardware constraints in designing DL applications is becoming increasingly important to build sustainable AI models, allow their deployments in resource-constrained edge devices, and reduce power consumption in large data centers. The standard hardware constraints of target hardware where the DL application is deployed are latency, memory occupancy, and energy consumption.

Hardware-aware NAS (HW-NAS) [2] addresses the above-mentioned limitations by including hardware constraints in the NAS search and optimization objectives to find efficient DL architectures. HW-NAS is comprised of three components: ❶ **the search space** which defines the types of DL architectures and how to construct them, ❷ **the search algorithm**, a multi-objective optimization strategy such as evolutionary algorithms or simulated annealing and ❸ **the evaluation method** where DL performance and efficiency, such as the accuracy and the hardware metrics, are computed on the target platform. In conventional NAS (figure 1.A), accuracy is the single objective that the search thrives on maximizing. Thus, the search algorithm only needs to evaluate the accuracy of each sampled architecture while exploring the search space to find the best architecture. On the other hand, HW-NAS (figure 1.B) is formulated as a multi-objective optimization problem, aiming to optimize two or more conflicting objectives, such as maximizing the accuracy of architecture and minimizing its inference latency, memory occupation and energy consumption. Dealing with multi-objective optimization becomes especially important in deploying DL applications on edge platforms. In a multi-objective optimization, the result obtained from the search algorithm is often not a single solution but a set of solutions. These solutions are called *dominant solutions* because they dominate all other solutions with respect to the trade-offs between the targeted objectives. A formal definition of dominant solutions is given in section 2. In the case of HW-NAS, the optimization result is a set of architectures with the best objectives' trade-off (figure 1.B). Formally, the set of best solutions is represented by a *Pareto front* (see section 2.1).

NAS algorithms train multiple DL architectures to adjust the exploration of a huge search space. This requires many hours/days of datacenter-scale computational resources. This time complexity is exacerbated in the case of HW-NAS multi-objective assessments, as additional evaluations are needed for each objective or hardware constraint on the target platform. To address this problem, researchers have proposed surrogate-assisted evaluation methods [16, 33]. Surrogate models use

[2]In the rest of the paper, we will use the term "architecture" to refer to "DL model architecture"

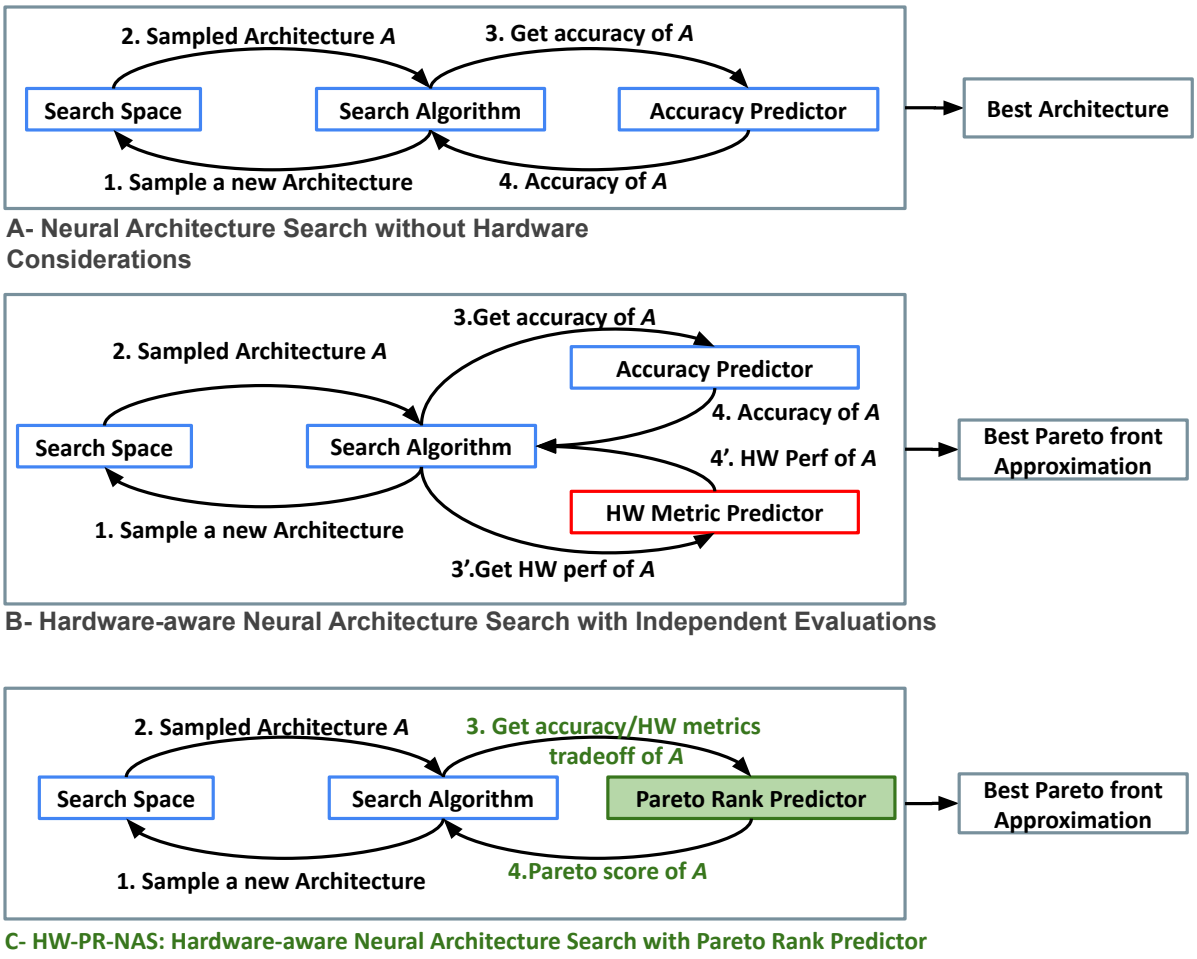

Fig. 1. Simplified illustration of using HW-PR-NAS in a NAS process. *HW Perf* means the Hardware performance of the architecture such as latency, power, etc.

analytical or ML-based algorithms that quickly estimate the performance of a sampled architecture without training it. Existing HW-NAS approaches [2] rely on the use of different surrogate-assisted evaluations, whereby each objective is assigned a surrogate, trained independently (figure 1.B). However, this introduces false dominant solutions as each surrogate model brings its share of approximation error and could lead to search inefficiencies and falling into local optimum (figures 2.a and 2.b).

Learning-to-rank theory [4, 33] has been used to improve the surrogate model evaluation performance. This was motivated by the following observation: *it is more important to rank a sampled architecture relatively to other architectures throughout the NAS process than to compute its exact accuracy*. Rank-preserving surrogate models significantly reduce the time complexity of NAS while enhancing the exploration path. However, in the multi-objective context, training each surrogate model independently cannot preserve the *Pareto rank* of the architectures, as illustrated in figure 2.

To speed up the exploration while preserving the ranking and avoiding conflicts between the surrogate models, we propose **HW-PR-NAS**, short for **Hardware-aware Pareto-Ranking NAS**. HW-PR-NAS is a unified surrogate model trained to simultaneously address multiple objectives in HW-NAS (figure 1.C). The contributions of the paper are summarized as follows:

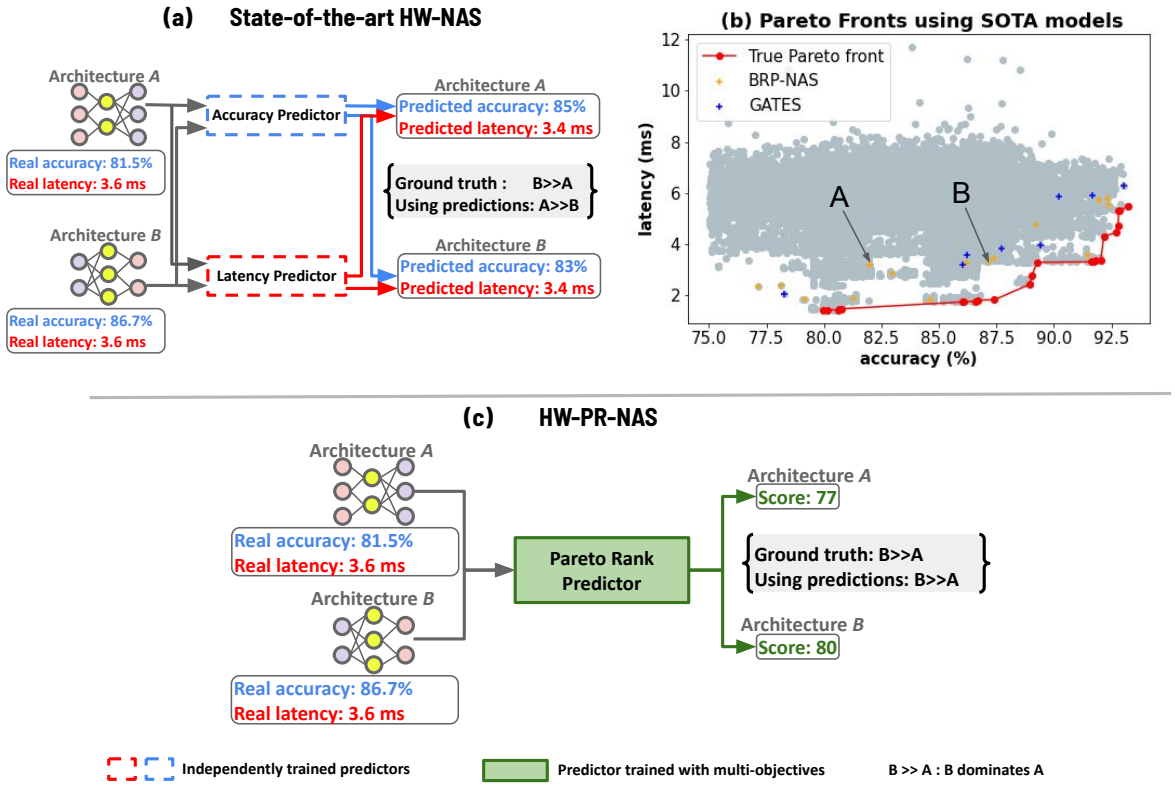

Fig. 2. This figure illustrates the limitation of state-of-the-art surrogate models alleviated by HW-PR-NAS. a) and b) illustrate how two independently trained predictors exacerbate the dominance error and the results obtained using GATES and BRP-NAS. c) illustrates how we solve this issue by building a single surrogate model.

(1) We introduce a **flexible and general architecture representation** which allows generalizing the surrogate model to include new hardware and optimization objectives without incurring additional training costs.
(2) We propose **a novel training methodology for multi-objective HW-NAS surrogate models**. Our surrogate model is trained using a novel ranking loss technique. The goal is to rank the architectures from dominant to non-dominant ones by assigning high scores to the dominant ones. These scores are called *Pareto scores*.

Our approach has been evaluated on seven edge hardware platforms, including ASICs, FPGAs, GPUs and multi-cores for multiple DL tasks, including image classification on CIFAR-10 and ImageNet and keywords spotting on Google Speech Commands. Experimental results demonstrate up to 2.5x speedup while guaranteeing that the search ends near the true Pareto front. We have evaluated HW-PR-NAS in the context of edge computing but our surrogate model's approach can be adapted to other platforms such as HPC or cloud systems.

Preliminary results show that using HW-PR-NAS is more efficient than using several independent surrogate models as it reduces the search time and improves the quality of the Pareto approximation.

The rest of this paper is organized as follows. Section 2 provides the relevant background. Section 3 discusses related work. Approach and methodology are described in Section 4. In Section 5, we

validate the proposed methodology by comparing our Pareto front approximations with state-of-the-art surrogate models, namely, GATES [33], and BRP-NAS [16]. Section 6 concludes the paper and discusses existing challenges and future research directions.

To allow a broad utilization of our work by the scientific community, we made the code and supplementary results available in a GitHub repository[3].

2 BACKGROUND

2.1 Multi-objective Optimization

Multi-objective optimization [31] deals with the problem of optimizing multiple objective functions simultaneously. Equation 1 formulates a multi-objective minimization problem, where A is the set of all the solutions, α is one solution and f_i with $i \in [1, \ldots, n]$ are the objective functions.

$$\min_{\alpha \in A} f_1(\alpha), \ldots, f_n(\alpha) \tag{1}$$

There is no single solution to these problems since the objectives often conflict. This means that we cannot minimize one objective without increasing another. Instead, the result of the optimization search is a set of dominant solutions called the *Pareto front*. In a two-objectives minimization problem, dominance is defined as: *If* s_1 *and* s_2 *denote two solutions,* s_1 ***dominates*** s_2 $(s_1 \succ s_2)$ *if and only if* $\forall i \ f_i(s_1) \leq f_i(s_2)$ *AND* $\exists j \ f_j(s_1) < f_j(s_2)$.

There is plenty of optimization strategies that address multi-objective problems, mainly based on meta-heuristics. One commonly used multi-objective strategy in the literature is the evolutionary algorithm [37]. The quality of the multi-objective search is usually assessed using the hypervolume indicator [17]. This metric computes the area of the objective space covered by the Pareto front approximation, i.e. the search result. The hypervolume, I_h, is bounded by the true Pareto front as a superior bound and a reference point as a minimum bound. Maximizing the hypervolume improves the Pareto front approximation and finds better solutions.

In our experiments, for the sake of clarity, we use the normalized hypervolume, which is computed with $I_h(\text{Pareto front approximation})/I_h(\text{true Pareto front})$. The closest to 1 is the normalized hypervolume, the better it is. However, if the search space is too big, we cannot compute the true Pareto front. Only the hypervolume of the Pareto front approximation is given.

2.2 Hardware-aware Neural Architecture Search (HW-NAS)

HW-NAS achieved promising results [7, 38] by thoroughly defining different search spaces and selecting an adequate search strategy. It refers to automatically finding the most efficient DL architecture for a specific dataset, task, and target hardware platform. HW-NAS approaches often employ black-box optimization methods such as evolutionary algorithms [13, 33], reinforcement learning [1], and Bayesian optimization [47].

The optimization problem is cast as: ⓐ A single objective function using scalarization such as a weighted sum of the objectives, i.e., task-specific performance and hardware efficiency. The weights are usually fixed via empirical testing. In this case, the result is a single architecture that maximizes the objective. ⓑ A pure multi-objective optimization where the result is a set of architectures representing the Pareto front. In formula 1, A refers to the architecture search space, α denotes a sampled architecture and f_i denotes the function that quantifies the performance metric i, where i may represent the accuracy, latency, energy consumption, or memory occupancy, etc.

During the search, the objectives are computed for each architecture. Because the training of a single architecture requires about 2 hours, the evaluation component of HW-NAS became the

[3]https://github.com/IHIaadj/HW-PR-NAS

bottleneck. For instance, MNASNet [38] needs more than 48 days on 64 TPUv2 devices to find the most efficient architecture within their search space. Evaluation methods quickly evolved into estimation strategies. The estimators are referred to as *Surrogate models* in this paper. Indeed, many techniques have been proposed to approximate the accuracy and hardware efficiency instead of training and running inference on the target hardware as described in the next section.

3 RELATED WORK

Table 1 illustrates the different state-of-the-art surrogate models used in HW-NAS to estimate the accuracy and latency.

Surrogate Model	Objective	Encoding	Loss	Dataset Size	Ranking
GATES [33]	Accuracy	GCN	Hinge Pair-wise	7318	yes
BRP-NAS [16]	Accuracy Latency	GCN GCN	MSE KL Div	900	no
ProxylessNAS [7]	Latency	AF	RMSE	5000	no
LRLC [44]	Accuracy	LSTM	Logistic Loss	1000	yes

Table 1. State-of-the-art surrogate models used for HW-NAS. AF stands for architecture features such as the number of convolutions and depth.

Below, we detail these techniques and explain how other hardware objectives, such as latency and energy consumption, are evaluated.

3.1 Accuracy Surrogate Models

Accuracy evaluation is the most time-consuming part of the search. Several approaches [16, 33, 44] propose ML-based surrogate models to predict the architecture's accuracy. We can distinguish two main categories according to the input of the surrogate model:

(1) ***Architecture Encoding***. The straightforward method involves extracting the architecture's features and then training an ML-based model to predict the accuracy of the architecture. GATES [33] and BRP-NAS [16] rely on a graph-based encoding that uses a Graph Convolution Network (GCN). Each architecture can be represented as a Directed Acyclic Graph (DAG), where the nodes are the input/intermediate/output data, and the edges are the operations, e.g., convolutions, pooling and attention. This makes GCN suitable for encoding an architecture's connections and operations.
Other methods [25, 27] use LSTMs to encode the architectural features, which necessitate the string representation of the architecture. In a smaller search space, FENAS [36] divides the architecture according to the position of the down-sampling operations. Then, it represents each block with the set of possible operations. They use random forest to implement the regression and predict the accuracy.
HAGCNN [41] uses a binary-based encoding dedicated to genetic search. Each operation is assigned a code. For example, the convolution 3x3 is assigned the "011" code. Then, they encode the architecture with a vector corresponding to the different operations it contains. During the search, they train the entire population with a different number of epochs according to the accuracies obtained so far.

(2) ***Learning Curves***. The learning curve is the loss obtained after training the architecture for a few epochs. We extrapolate or predict the accuracy in later epochs using these loss values. In [44], the authors use the results of training the model for 30 epochs, the architecture

encoding, and the dataset characteristics to score the architectures. This scoring is learned using the pairwise logistic loss to predict which of two architectures is the best.

A more detailed comparison of accuracy estimation methods can be found in [43].

3.2 Hardware Efficiency Surrogate Models

Latency is the most evaluated hardware metric in NAS. Several works in the literature have proposed latency predictors. We can classify them into two categories:

(1) ***Layer-wise predictor***. In a preliminary phase, we estimate the latency of each possible layer in the search space. We can either store the approximated latencies in a lookup table (LUT) [6] or develop analytical functions that, according to the layer's hyperparameters, estimate its latency. The end-to-end latency is predicted by summing up all the layers' latency values. This layer-wise method has several limitations for NAS performance prediction [2, 16].
(2) ***End-to-end predictor***. Similar to the conventional NAS, HW-NAS resorts to ML-based models to predict the latency. ProxylessNAS [7] uses a surrogate model based on manually extracted features such as the type of the operator, input and output feature map size, and kernel sizes. BRP-NAS [16], on the other hand, uses a GCN to encode the architecture and train the final fully-connected layer to regress the latency of the model.

For other hardware efficiency metrics such as energy consumption and memory occupation, most of the works [18, 32] in the literature use analytical models or lookup tables.

Novelty Statement. To the best of our knowledge, this paper is the first work that builds a single surrogate model for Pareto ranking task-specific performance and hardware efficiency. Our approach is motivated by the fact that using multiple independently trained surrogate models for each objective only delivers sub-optimal results, as each surrogate model will bring its share of error. Instead, we train our surrogate model to predict the Pareto rank as explained in section 4. A single surrogate model for Pareto ranking provides a better Pareto front estimation and speeds up the exploration. For the sake of clarity, we focus on a two-objective optimization; accuracy and latency. We then explain how we can generalize our surrogate model to add more objectives in section 5.5.

4 PROPOSED APPROACH: HW-PR-NAS

Figure 3 shows an overview of HW-PR-NAS, which is composed of two main components: ❶ **Encoding Scheme** and ❷ **Pareto Rank Predictor**.

Each architecture is encoded into a unique vector and then passed to the Pareto Rank Predictor in the **Encoding Scheme**. The **Pareto Rank Predictor** uses the encoded architecture to predict its Pareto Score (see equation 7) and adjusts the prediction based on the Pareto Ranking Loss. The Pareto Score, a value between $0 and 1$, is the output of our predictor. We use a listwise Pareto ranking loss to force the Pareto Score to be correlated with the Pareto ranks.

4.1 Encoding Schemes

Definitions. In this paper, we use the following terms with their corresponding definitions:

- *Representation:* is the format in which the architecture is stored.
- *Encoding:* is the process of turning the architecture representation into a numerical vector. The surrogate model can then use this vector to predict its rank.
- *Encoding scheme:* is the methodology used to encode an architecture.
- *Encoder:* is a function that takes as input an architecture and returns a vector of numbers, i.e., applies the encoding process.

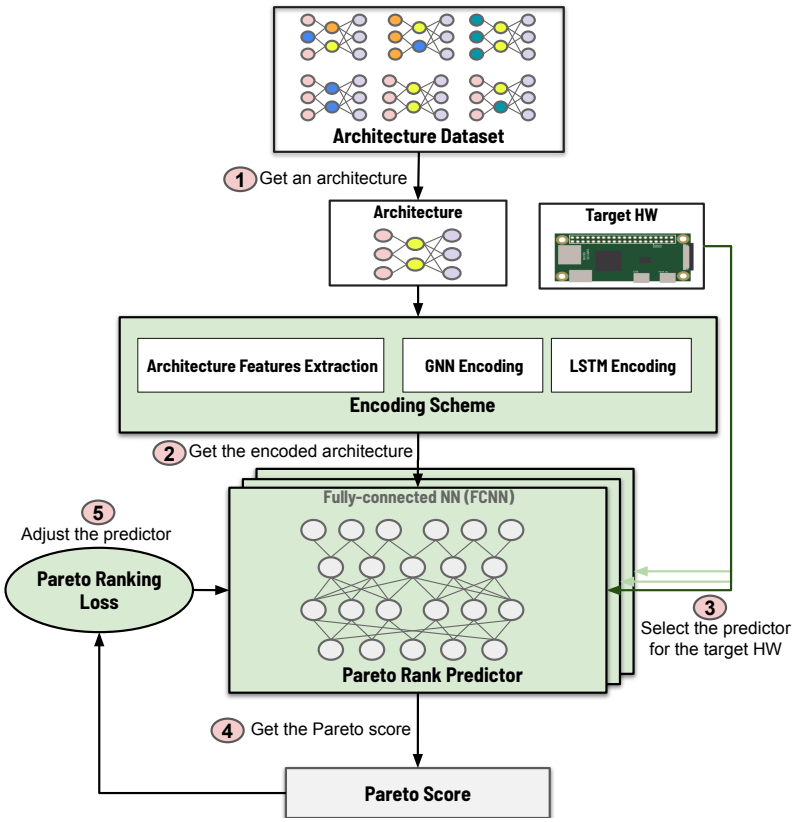

Fig. 3. General Overview of HW-PR-NAS

- *Pareto Rank Predictor:* is last part of the model architecture specialized in predicting the final score of the sampled architecture (see figure 3).

To achieve a robust encoding capable of representing most of the key architectural features, HW-PR-NAS combines several encoding schemes (see figure 3). Each architecture is described using two different representations: a *Graph Representation*, which uses Directed Acyclic Graphs (DAG), and a *String Representation*, which uses discrete tokens that express the NN layers. For example using "conv_3x3" to express a 3x3 convolution operation. We use two encoders to represent each architecture accurately. Both representations allow using different encoding schemes. Each encoder can be represented as a function E formulated as follows:

$$E : A \rightarrow \xi \tag{2}$$

A denotes the search space, and ξ denotes the set of encoding vectors. The encoder E takes an architecture's representation as input and maps it into a continuous space ξ. The encoding result is the input of the predictor.

In our approach, three encoding schemes have been selected depending on their representation capabilities and the literature review (see table 1):

(1) **Architecture Features Extraction.** From each architecture, we extract several Architecture Features (AF): number of FLOPs, number of parameters, number of convolutions, input size, architecture's depth, first and last channel size, and number of down-sampling.
(2) **GCN Encoding.** To efficiently encode the connections between the architecture's operations, we apply a GCN encoding. Each architecture is encoded into its adjacency matrix and

	Hyperparameter	Value
GCN Encoding	Number of layers	2
	hidden depth	128
	hidden dimension	1
	FC dimension	32
LSTM Encoding	Number of layers	2
	Hidden units	[32, 64]
	FC dimension	32
	recurrent dropout	0.2
Decoder	Number of layers	3
	Hidden units	[32,32]

Table 2. Hyperparameters associated with GCN and LSTM encodings and the decoder used to train them.

operation vector. It is then passed to a Graph Convolution Neural Network (GCN) [20] to generate the encoding. The output is passed to a dense layer to reduce its dimensionality.

(3) **LSTM Encoding.** To represent the sequential behaviour of the architecture, we use an LSTM encoding scheme. We pass the architecture's string representation through an embedding layer and an LSTM model. We then reduce the dimensionality of the last vector by passing it to a dense layer.

The resulting encoding is a vector that concatenates the AFs to ensure that each architecture in the search space has a unique and general representation that can handle different tasks [28] and objectives. The hyperparameters describing the implementation used for the GCN and LSTM encodings are listed in table 2.

Using a decoder module, the encoder is trained independently from the Pareto rank predictor. The decoder takes the concatenated version of the three encoding schemes and recreates the representation of the architecture. We set the decoder's architecture to be a 4-layers LSTM. In addition, we leverage the attention mechanism to make decoding easier. The encoder-decoder model is trained with the cross entropy loss. Equation 3 formulates the cross entropy loss, denoted as L_{ED}, where $output\_size$ changes according to the string representation of the architecture, y and $\hat{y}$ correspond to the predicted operation and the true operation respectively. This training methodology allows the architecture encoding to be hardware-agnostic.

$$L_{ED} = -\sum_{i=1}^{output\_size} y_i * log(\hat{y}_i) \tag{3}$$

The preliminary analysis results in figure 4 validate the premise that different encodings are suitable for different predictions in the case of NAS objectives. Figure 4 shows the results obtained after training the accuracy and latency predictors with different encoding schemes. Each predictor is trained independently. Using Kendal Tau [34], we measure the similarity of the architectures' rankings between the ground truth and the tested predictors. Accuracy predictors are sensible to the types of operators and connections in a DL architecture. When using only the AF, we observe a small correlation (0.61) between the selected features and the accuracy, resulting in poor performance predictions. The best predictor is obtained using a combination of GCN encodings, which encodes the connections, node operation, and AF. For latency prediction, results show that the LSTM encoding is better suited. An intuitive reason is that the sequential nature of the operations to compute the latency is better represented in a sequence string format. The last two columns of the

figure show the results of the concatenation, which outperforms other representations as it holds all the features required to predict the different objectives.

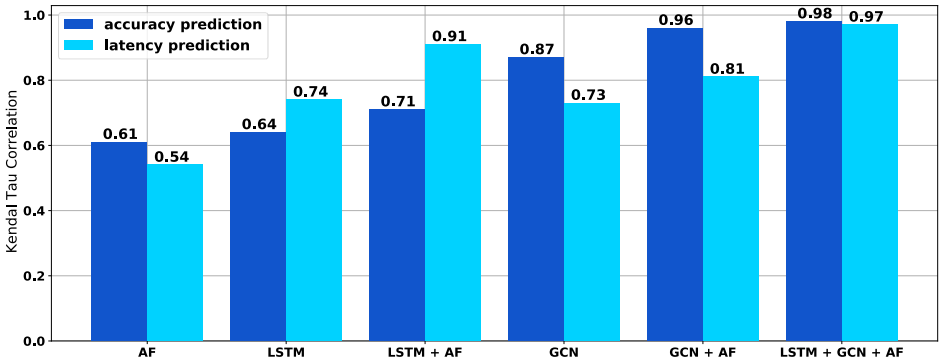

Fig. 4. Results of different encoding schemes for accuracy and latency predictions on NAS-Bench-201 and FBNet. AF refers to Architecture Features. LSTM refers to Long Short-Term Memory neural network. GCN refers to Graph Convolutional Networks.

These results were obtained with a fixed Pareto Rank predictor architecture. We used a fully-connected neural network (FCNN). Table 3 shows the results of modifying the final predictor on the latency and accuracy predictions. While we achieve a slightly better correlation using XGBoost on the accuracy, we prefer to use a 3-layer FCNN for both objectives to ease the generalization and flexibility to multiple hardware platforms.

	Accuracy		Latency	
	RMSE	KT Corr	RMSE	KT Corr
3-layer FCNN	4.88	0.924	3.238	**0.8817**
XGBoost [8]	3.12	**0.931**	3.216	0.8742
LGBoost [19]	3.58	0.864	3.058	0.8247

Table 3. Results of different regressors on NAS-Bench-201. KT Corr stands for Kendal Tau Correlation.

4.2 Pareto Ranking Predictor

HW-PR-NAS is trained to predict the Pareto front ranks of an architecture for multiple objectives simultaneously on different hardware platforms. The predictor uses three fully-connected layers. Due to the hardware diversity illustrated in table 4, the predictor is trained on each HW platform. Prior works [2] demonstrated that the best architecture in one platform is not necessarily the best in another. Therefore, the Pareto fronts differ from one HW platform to another.

HW-PR-NAS predictor architecture is the same across the different HW platforms. The only difference is the weights used in the fully-connected layers. The HW platform identifier (*Target HW* in figure 3) is used as an index to point to the corresponding predictor's weights.

To train this Pareto ranking predictor, we define a novel *listwise* loss function to predict the Pareto ranks.

Pareto ranks definition. In a multi-objective NAS problem, the solution is a set of N architectures $S = s_1, s_2, ..., s_N$. These architectures may be sorted by their Pareto front rank K. The true Pareto front is denoted as F_1 where the rank of each architecture within this front is 1. An architecture is in

the true Pareto front if and only if it dominates all other architectures in the search space. According to this definition, we can define the Pareto front ranked 2, F_2, as the set of all architectures that dominate all other architectures in the space except the ones in F_1. Formally, the rank K is the number of Pareto fronts we can have by successively solving the problem for $S - \bigcup_{s_i \in F_k \wedge k<K}$, i.e., the top dominant architectures are removed from the search space each time.

Theoretically, the sorting is done by following these conditions:

$$\forall s_i, s_j \in F_k,\ s_i \not\succ s_j \wedge\ s_j \not\succ s_i \tag{4}$$

$$\forall s_i \in F_{k+1}\ \forall s_j \in F_k,\ s_i \not\succ s_j \tag{5}$$

$$\forall s_i \in F_{k+1}\ \exists s_j \in F_k,\ s_j \succ s_i \tag{6}$$

Equation 4 formulates that for all the architectures with the same Pareto rank, no one dominates another. Equation 5 formulates that any architecture with a pareto rank $k+1$ cannot dominate any architecture with a pareto rank k. Equation 6 formulates that for each architecture with a Pareto rank $k+1$, at least one architecture with a Pareto rank k dominates it.

Pareto ranking loss definition. Our predictor takes an architecture as input and outputs a score. This score is adjusted according to the Pareto rank. The loss function aims to keep the predictor's outputs; scores $f(a)$, where a is the input architecture, correlated to the actual Pareto rank of the given architecture.

The scores are then passed to a softmax function to get the probability of ranking architecture a. The final output is formulated as follows:

$$out(a) = \frac{\exp f(a)}{\sum_{a \in B} \exp f(a)} \tag{7}$$

In this equation, B denotes the set of architectures within the batch, while $|B|$ denotes its size. We then design a listwise ranking loss by computing the sum of the negative likelihood values of each batch's output:

$$L(B) = \sum_{i=1}^{|B|} \{-out(a^{(i),B}) + log \sum_{j=i}^{|B|} exp(out(a^{(j),B})\} \tag{8}$$

$a^{(i),B}$ denotes i-th Pareto-ranked architecture in subset B. This loss function computes the probability of a given permutation to be the best, i.e., if the batch contains three architectures a_1, a_2, a_3 ranked (1, 2, 3) respectively. The loss function encourages the surrogate model to give higher values to architecture a_1 then a_2 and finally a_3. We compute the negative likelihood of each architecture in the batch being correctly ranked.

Training procedure. To train the HW-PR-NAS predictor with two objectives, the accuracy and latency of a model, we apply the following steps:

(1) We build a ground truth dataset of architectures and their Pareto ranks. We randomly extract architectures from NAS-Bench-201 and FBNet using Latin Hypercube Sampling [29]. The batches are shuffled after each epoch. The two benchmarks already give the accuracy and latency results. Thus the dataset creation is not computationally expensive. However, if one uses a new search space, the dataset creation will require at least the training time of 500 architectures.

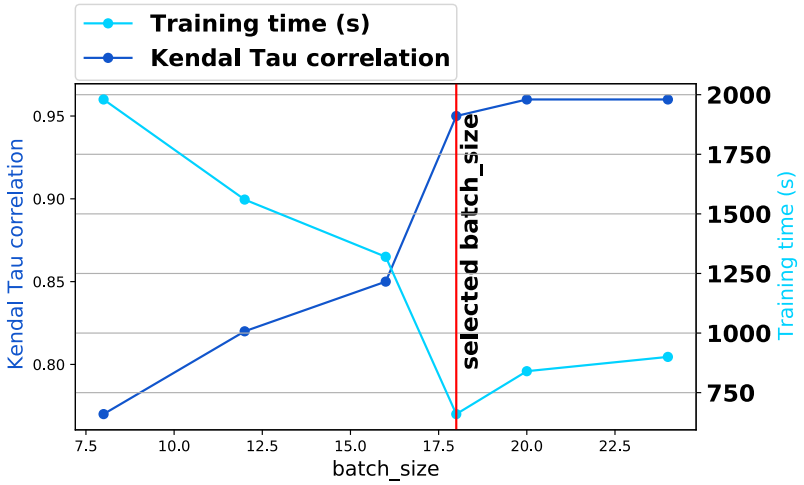

Fig. 5. Performance of the Pareto rank predictor using different batch_size values during training.

(2) We iteratively compute the ground truth of the different Pareto ranks between the architectures within each batch using the actual accuracy and latency values. Two architectures with close Pareto score means that both have the same rank.
(3) We calculate the loss between the predicted scores and the ground truth computed ranks.
(4) Using this loss function, the scores of the architectures within the same Pareto front will be close to each other, which helps us extract the final Pareto approximation.

Algorithm 1: Training methodology of the rank predictor component.

Data: *benchmark*: Benchmark (α: architecture, acc: accuracy, l: latency)
Result: Trained Surrogate Model
$D \leftarrow Sample($ benchmark ,dataset_size)
model $\leftarrow FCNN$
initialize model
for *epoch* < *max_epochs* **do**
  batches = generate_random_batches(D)
  for *B in batches* **do**
    R $\leftarrow Compute\_Pareto\_Ranks$(B)
    P $\leftarrow model$(B)
    $loss \leftarrow NLL(R, P)$
    backpropagate *loss* and adjust the weights of the model

The most important hyperparameter of this training methodology that needs to be tuned is the batch_size. Figure 5 shows the empirical experiment done to select the *batch_size*. We set the batch_size to 18 as it is, empirically, the best trade-off between training time and accuracy of the surrogate model. The accuracy of the surrogate model is represented by the Kendal tau correlation between the predicted scores and the correct Pareto ranks. This value can vary from one dataset to another. The hyperparameter tuning of the batch_size takes ~1h for a full sweep of 6 values in this range: [8, 12, 16, 18, 20, 24].