# SEval-NAS: A Search-Agnostic Evaluation for Neural Architecture Search

Atah Nuh Mih
Analytics Everywhere Lab,
University of New Brunswick
Fredericton, Canada
atah.mih@unb.ca

Jianzhou Wang
Analytics Everywhere Lab,
University of New Brunswick
Fredericton, Canada
maxwell.wang@unb.ca

Truong Thanh Hung Nguyen
Analytics Everywhere Lab,
University of New Brunswick
Fredericton, Canada
hung.ntt@unb.ca

Hung Cao
Analytics Everywhere Lab,
University of New Brunswick
Fredericton, Canada
hcao3@unb.ca

## Abstract

Neural architecture search (NAS) algorithms have automated the discovery of new neural networks by generating candidate architectures that meet desired criteria. Evaluating these candidate architectures is often hardcoded into the algorithms, making it challenging to adopt new evaluation criteria without significantly changing the algorithm. This inflexibility is especially evident in hardware-aware NASs whose search objectives are tailored for hardware such as edge devices. To overcome this challenge, we propose a metric-evaluation mechanism called SEval-NAS that converts neural networks to strings, obtains their vector representation, and predicts the evaluation metric. We experimented on two NAS benchmarks: NATS-Bench and HW-NAS-Bench, evaluating the neural architectures on accuracy, latency, and memory. By comparing Kendall's $\tau$ correlation coefficients in these experiments, the results showed that latency and memory predictions had stronger correlations than accuracy, indicating SEval-NAS' suitability as a hardware cost predictor. We further investigated the feasibility of integrating SEval-NAS in an existing NAS algorithm by evaluating candidate architectures in FreeREA on metrics not originally included. The results showed that our method successfully ranked architectures generated by FreeREA, did not drastically affect search time, and did not require significant changes to the search algorithm.

## CCS Concepts

• **Computing methodologies** → **Neural networks**; *Search methodologies*; • **Theory of computation** → **Network optimization**.

## Keywords

Neural architecture search, network optimization

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*SAC'26, Thessaloniki, Greece*
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-X-XXXX-XXXX-X/26/03
https://doi.org/XXXXXXX.XXXXXXX

**ACM Reference Format:**
Atah Nuh Mih, Jianzhou Wang, Truong Thanh Hung Nguyen, and Hung Cao. 2026. SEval-NAS: A Search-Agnostic Evaluation for Neural Architecture Search. In *Proceedings of The 41st ACM/SIGAPP Symposium on Applied Computing (SAC'26).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

Neural architecture search (NAS) was developed to automate the design of neural networks (NNs), addressing the knowledge gap required to design these networks manually. Traditional NAS [35] used reinforcement learning (RL) to generate variable-length strings representing architectures, achieving state-of-the-art performance but at high computational cost. This shortcoming prompted research into more efficient search methods, such as a cell-based search space with learnable transferable architectures [34], Efficient NAS (ENAS) [23], Progressive NAS (PNAS) [17], and Regularized Evolution for Image Classifier Architecture Search [25]. In general, NAS is divided into five major components: (1) the search space containing predefined operation sets; (2) the controller that determines how neural architectures are generated; (3) the candidate architecture(s) generated; (4) the evaluation phase with a strategy for assessing architectures feasibility; and (5) the optimal architecture(s) that satisfy search objectives.

The evaluation phase is one of the most critical steps in NAS. It evaluates the performance of candidate architecture(s) for desired objectives and guides the search algorithm towards an optimal architecture [5]. Depending on the evaluation method used, this phase could pose a significant search cost to the NAS approach since all the candidate architectures are trained and tested to assess their performance [2]. For example, [34] trained each candidate architecture till convergence on proxy data to obtain its evaluation metrics, resulting in a search cost of 22,400 GPU hours. The shortcomings of full training motivated incomplete training to accelerate the ranking of candidate architectures, thereby reducing search costs [26]. Another challenge with existing NAS approaches is the inflexibility of accommodating new evaluation metrics. Different NAS approaches aim to find architectures that satisfy different performance criteria. For example, various hardware-aware NAS

(HW-NAS) algorithms include hardware cost metrics such as latency [29, 30, 10] and memory [16] to select candidate architectures suitable for hardware platforms such as edge devices. Thus, the evaluation criteria are hardcoded into the search design of NAS algorithms, making it challenging to adopt new evaluation metrics without re-designing the search. Therefore, an evaluation mechanism that is independent of the search algorithm and can be flexibly adapted to any NAS approach is necessary.

To address these challenges, we propose a search-agnostic evaluation approach called SEval-NAS. It converts any NN into its string representation, encodes the string to obtain the vector embedding, and predicts the evaluation metrics. This is based on the premise that NNs' performance reflects the structural dependencies of their internal operations (e.g., type of convolution, number of filters, type of activations). Extracting this structural information can, therefore, help predict their given performance. We show that SEval-NAS supports different types of metrics and evaluation objectives and can be directly applied to an existing NAS method with minimal changes to the search algorithm and without significantly affecting the search. We experiment on two NAS benchmarks: NATS-Bench [7] and HW-NAS-Bench [15] for accuracy, latency, and memory, and further assess how our method affects a NAS algorithm (i.e., FreeREA [2]).

In summary, this work presents the following contributions:

- A network-to-string conversion mechanism that traverses the autograd graph of any NN and generates its textual representation, making it adaptive to all types of NNs.
- An encoder-predictor network (i.e., an evaluator) that extracts meaningful relationships between the strings and their evaluation metric. This network can be designed to include any evaluation metric (notably hardware costs) and any number of evaluation objectives.
- SEval-NAS that is independent of the NAS algorithm and combines the network-to-string conversion mechanism and the evaluator to evaluate candidate architectures in NAS.
- An ablation study of three different encoder/decoder models (T5-small, T5-base, and T5-large) in SEval-NAS on NAS benchmarks.

## 2 Literature Review

This section discusses the most relevant works related to training-free NAS, HW-NAS, and NAS as a string search problem, highlighting the gaps that motivated our approach.

### 2.1 Training-Free NAS

The costly evaluation of candidate architectures has motivated the development of training-free metrics that evaluate candidate architectures and reduce the search time of NAS. [1] proposed regression models to predict the final performance of models from learning curve trajectories based on features obtained from the neural architectures, hyperparameters, and time-series measurements. [21] proposed NASWOT, which examines the correlation of activations between data points in untrained NNs and scores networks based on the binary codes corresponding to this correlation. [4] proposed TE-NAS, a training-free NAS that analyzes the spectrum of the neural tangent kernel (NTK) and the number of linear regions to rank candidate architectures. While these methods successfully address the evaluation of candidate architectures, they solely focus on accuracy as their performance metrics. We extend beyond accuracy alone by including hardware metrics to assess the suitability of architectures for different computing environments.

Hardware cost predictors have been developed, such as a nn-Meter [32], a latency predictor for edge devices; and a GPU estimator for deep learning models [9]. Integrating different single-purpose cost predictors will increase the complexity of the design, so we propose a multi-purpose cost estimator that can incorporate different hardware costs.

### 2.2 Hardware-Aware NAS

The search for good neural architectures extends beyond just high-accuracy networks. In cases where the hardware environment is crucial, cost metrics must be considered when evaluating the NNs. This requirement gave rise to HW-NAS, which includes a hardware cost metric and test accuracy in evaluating candidate architectures. Several HW-NAS approaches have been proposed, such as SqueezeNext [11], IRLAS [12], and FB-Net [30]. These works only provide the optimal architectures obtained from the search, leaving a question about their performance if the search were to be evaluated differently.

Hardware-aware NAS equally targets edge devices because their hardware environments require NNs suitable for their resource constraints. Several NAS methods have been developed for edge devices [14, 29, 20, 28]. While these works have been successful in searching optimized NNs, they are usually designed to satisfy a single hardware cost metric. Latency has often been used as the evaluation metric in hardware-aware NAS [19, 33, 10, 16], whereas few works include multiple cost metrics in their design [27].

These NAS methods are generally multi-objective and aim to satisfy more than one primary objective pre-defined while designing the search. Our approach provides an easy integration of a cost evaluator for any desired objective (e.g., accuracy, latency, and memory) and several objectives (e.g., single or bi-objectives).

### 2.3 NAS as a String Search Problem

NAS as a string search problem was proposed in the earliest NAS work [34], where a variable-length string specifies the search space, and a recurrent NN (i.e., controller) is used to generate such a string using RL. GeNet [31] adopted a similar approach, representing network structures as fixed-length binary strings and using genetic algorithms to generate new architectures. [18] proposed Neural Architecture Optimization (NAO) to find candidate networks using an encoder, a predictor, and a decoder network to perform the search. The encoder maps the architectures to a continuous vector space, the predictor approximates the classification accuracy, and the decoder attempts to reconstruct the architecture. They directly optimize the predictor by searching the embedding of neural architectures to derive the best architectures. Like other NAS methods, the optimization is close-knit, with a prediction built into the search itself. Contrary to this, our prediction mechanism is *plug-and-play* and can complement existing NAS. We also expand the prediction to include hardware costs, which is crucial to HW-NAS. A more

recent EVOPROMPTING [3] uses language modeling and prompting for code-level NN generation. This approach, however, is very high-level as it generates programming code.

## 3 Methodology

Our proposed SEval-NAS framework is designed to evaluate neural architectures within a NAS pipeline by leveraging a formalized string-based representation and a predictive evaluation model. Let $\mathcal{A}$ denote the set of candidate neural architectures, where each architecture $a \in \mathcal{A}$ is characterized by its computational graph. The methodology transforms each architecture into a standardized string representation, which is subsequently processed by an evaluator to predict performance metrics. The predicted metrics guide the NAS controller in optimizing the search process. The framework consists of two primary components: (1) a network-to-string conversion mechanism and (2) an evaluator network for performance prediction. Fig. 1 provides a schematic of the proposed methodology, illustrating its integration within a NAS pipeline.

### 3.1 Net-to-String Conversion

The network-to-string conversion process maps a neural architecture $a \in \mathcal{A}$ to a string representation $s_a \in \mathcal{S}$, where $\mathcal{S}$ is the space of all possible string representations. Let $G_a = (V_a, E_a)$ represent the computational graph of architecture $a$ (generated during the forward pass), with vertices $V_a$ corresponding to operations (e.g., convolution, pooling, ReLU) and edges $E_a$ representing data flow between operations. The conversion function $f : \mathcal{A} \to \mathcal{S}$ traverses $G_a$ (breadth-first) to extract structural and operational details, yielding a string $s_a$ that encapsulates the architecture's configuration.

Formally, the conversion process is defined as: $s_a = f(G_a)$, where $f$ systematically traverses $V_a$ and $E_a$ to encode operations and their connectivity into a standardized format. The resulting string $s_a$ is tokenized into a sequence of tokens $T_a = \{t_1, t_2, \ldots, t_n\}$, where each token $t_i$ corresponds to a specific operation or parameter in the computational graph. This tokenization ensures a universal and consistent representation, enabling compatibility across diverse NAS tasks and datasets. The conversion is described in Algorithm 1.

**Algorithm 1** Network to String Conversion

**Input:** Neural Network: $net_a$; Input Tensor: $inp$
**Output:** Output token $T_a$

$G_a \longleftarrow net_a(inp)$ ▷ forward pass
$node = G_a(root)$ ▷ root node
$s_a = \emptyset$ ▷ output string
$seen = \emptyset$ ▷ keep track of traversal
$node\_id = 0$ ▷ node index
**function** GET_STRING(node)
  **if** $node$ in $seen$ **then return**
  **end if**
  $seen \longleftarrow node$
  $next\_nodes = node.next\_functions$ ▷ neighbouring vertices
  **for** $u$ in $next\_nodes$ **do**
    GET_STRING(u)
  **end for**
  $vars = node.variable$ ▷ get operations and variables
  $name = node.name$
  $s_a \longleftarrow name, node\_id, vars$
**end function**
$T_a = tokenize(s_a)$ ▷ tokenization with any desired method

### 3.2 Evaluator

The tokenized input is processed in the evaluator module to predict its performance metrics. The evaluator module, denoted as $\mathcal{E}$, predicts performance metrics for a given architecture based on its tokenized representation $T_a$. Let $\mathcal{M} = \{m_1, m_2, \ldots, m_k\}$ represent the set of target performance metrics (e.g., accuracy, latency, memory usage). The evaluator maps the tokenized input to a vector of predicted metrics: $\hat{m}_a = \mathcal{E}(T_a)$, where $\hat{m}_a = [\hat{m}_{a,1}, \hat{m}_{a,2}, \ldots, \hat{m}_{a,k}] \in \mathbb{R}^k$ denotes the predicted values for the $k$ metrics.

The module consists of two components: an encoder and a predictor.

(1) **Encoder**: The encoder extracts a high-dimensional vector representation of the architecture, capturing its structural and contextual information. It is represented by a function $g : \mathcal{T} \to \mathbb{R}^d$ that transforms the tokenized sequence $T_a$ into a high-dimensional embedding $e_a \in \mathbb{R}^d$, capturing structural and contextual features of the architecture:

$$e_a = g(T_a).$$

The encoder employs a transformer-based architecture to model dependencies within the token sequence, ensuring robust feature extraction.

(2) **Predictor**: A function $h : \mathbb{R}^d \to \mathbb{R}^k$ that maps the embedding $e_a$ to the predicted metrics:

$$\hat{m}_a = h(e_a).$$

The prediction layer is a fully connected neural network with $k$ output neurons, where the number of neurons corresponds to the number of target metrics. For single-objective prediction (e.g., latency), $k = 1$, while for multi-objective prediction (e.g., accuracy and latency), $k \geq 2$.

The evaluator is trained on a dataset $\mathcal{D} = \{(a_i, m_i)\}_{i=1}^N$, where $m_i \in \mathbb{R}^k$ are the true performance metrics for architecture $a_i$. The training objective minimizes the loss function:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \ell(\mathcal{E}(T_{a_i}), m_i),$$

where $\ell$ is a regression loss (e.g., mean squared error) that measures the discrepancy between predicted and true metrics.

### 3.3 Integration into NAS Pipeline

The SEval-NAS framework is integrated into a NAS pipeline by evaluating candidate architectures generated by the controller. Let $C$ denote the controller, which generates architectures $a \in \mathcal{A}$ based on a search strategy. The evaluator provides feedback in the form of predicted metrics $\hat{m}_a$, enabling the controller to optimize the search objective:

$$a^* = \arg\max_{a \in \mathcal{A}} u(\hat{m}_a),$$

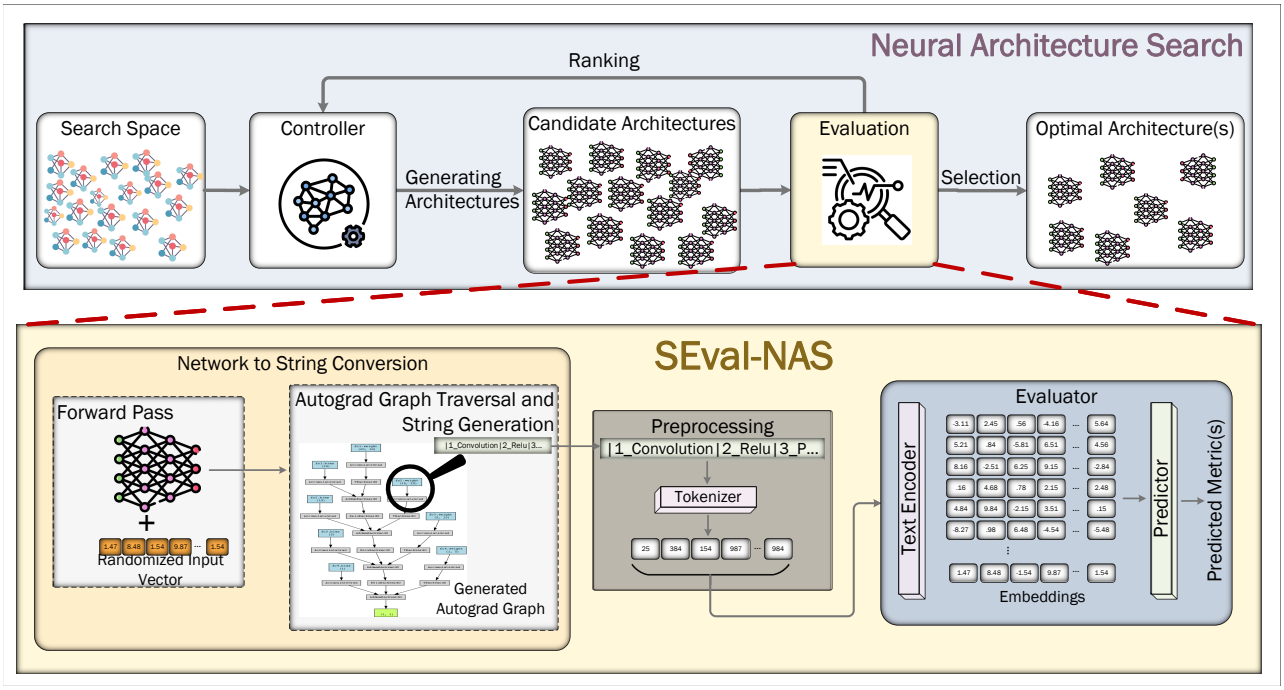



**Figure 1: Proposed SEval-NAS methodology and its integration in a NAS pipeline**

where $u : \mathbb{R}^k \to \mathbb{R}$ is a utility function that aggregates the predicted metrics (e.g., a weighted sum for multi-objective optimization). The schematic of this integration is illustrated in Figure 1, highlighting the closed-loop interaction between the controller, candidate architectures, and the SEval-NAS evaluator.

This modular design ensures that SEval-NAS can be seamlessly incorporated into existing NAS frameworks, such as FreeREA, without requiring significant modifications to the search algorithm. The flexibility of the prediction layer allows adaptation to varying numbers of objectives, enhancing the applicability of SEval-NAS across diverse hardware and performance constraints.

## 4 Experiments and Results

We evaluated the effectiveness of SEval-NAS using two NAS benchmarks: NATSBench [7] and HW-NAS-Bench [15]. Our evaluation focused on how well our method's predictions correlated with actual performance metrics. Additionally, we demonstrate the adaptability of our approach by applying it to FreeREA. We run our experiments on a 13th Gen Intel(R) Core(TM) 9-13900K server equipped with NVIDIA GeForce RTX 4090.

### 4.1 Model Configuration

The evaluator in SEval-NAS is a transformer encoder whose input is the neural network's text representation and outputs the embedding into a regression head for prediction. Specifically, we use the encoder from the T5 transformer [24]. It consists of stacked layers containing a self-attention layer and a small feed-forward network, followed by layered normalization and a residual skip connection. Dropout is strategically applied to the feed-forward network, the skip connection, the attention weights, and the stack's input and output. We use three different sizes of T5 models :

- T5-small, which uses 8-headed attention, has only 6 layers each in the encoder and decoder, and has roughly 60 million parameters.
- T5-base, which uses 12-headed attention, has 12 layers each in the encoder and decoder, and has nearly 220 million parameters.
- T5-large, which uses 16-headed attention, has 24 layers each in the encoder and decoder, and has approximately 770 million parameters

The predictor is a single dense layer whose number of output neurons depends on the number of desired objectives.

### 4.2 Training

The evaluator (T5-small model) is trained to predict performance metrics on datasets containing NNs and their reported metrics. Other models (i.e., T5-base and T5-large) are trained and evaluated via an ablation study (see Appendix A). In NAS, these datasets exist as NAS benchmarks containing thousands of neural architectures and metrics such as accuracy, latency, and FLOPS obtained from training and inferencing those networks. Typically, these benchmarks exclude memory usage, prompting the need for additional profiling. To address this, we build a lookup table containing the peak memory usage using the built-in PyTorch memory profiling tool [1]. This tool measures the peak memory allocated to tensors during training, providing an accurate assessment of NNs' memory

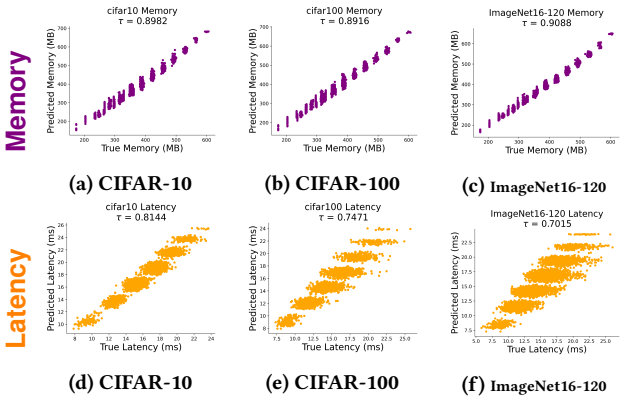
[1] torch.cuda.max_memory_allocated

footprint. The profiling is isolated from memory used by external factors such as libraries or system variables, ensuring precise measurement. We train our evaluator on two NAS benchmarks: NATS-Bench [7] and HW-NAS-Bench [15].

### 4.3 Experiment 1: Feasibility Testing (Evaluation on NATS-Bench)

NATS-Bench [7] is a unified benchmark dataset for searching on both architecture topology and size. It consists of 15,625 different architectures for the Topology Search Space (TSS) and 32,768 architectures for the Size Search Space (SSS) evaluated on CIFAR10, CIFAR100, and ImageNet16-120.

In the TSS, each architecture corresponds to a different cell represented as a densely connected directed acyclic graph (DAG) with four nodes and edges corresponding to operations from a predefined set of 5 operations. For each architecture in the TSS, the cells are stacked 5 times, with output channels set to 16, 32, and 64 for three stages. The search results in a search space containing 15,625 possible architectures configured for the image dataset considered (i.e., CIFAR-10, CIFAR-100 [13], and ImageNet16-120 [6]). The SSS searches for architectures by varying the number of channels in each layer (convolution, cell, or block). Each architecture consists of a stacked cell, and the number of channels in each layer is selected from the set $\{8, 16, 24, 32, 40, 48, 56, 64\}$, resulting in 32,768 architectures. For our experiment, we evaluate the TSS and SSS. In each search space, we separately train the evaluator on the CIFAR-10, CIFAR-100, and ImageNet16-120 architectures, configuring our network for both *(accuracy, memory)* and *(accuracy, latency)* bi-objective setups.

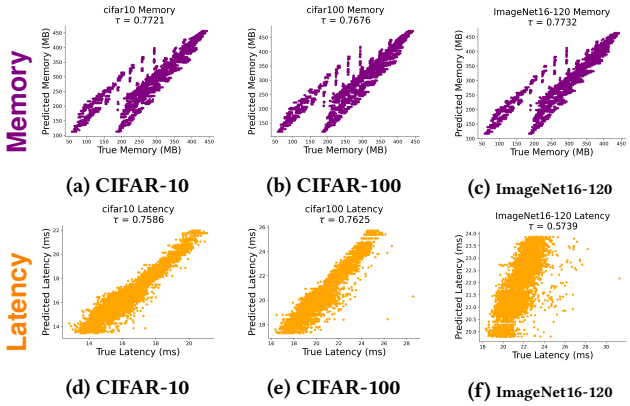



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet16-120

(d) CIFAR-10 (e) CIFAR-100 (f) ImageNet16-120

**Figure 2: Plots of predicted vs true hardware cost of NATS-Bench TSS architectures (using T5-small) for performance metrics reported on CIFAR-10, CIFAR-100, and ImageNet16-120 datasets. The strength of correlation increases as $\tau$ approaches 1.**

The results of the NATS-Bench TSS in Fig. 2 show a strong positive Kendall $\tau$ correlation between predicted and true values for hardware costs. Predicted memory usage aligns closely with the true values across all datasets. Similarly, latency predictions exhibit a high correlation for CIFAR-10 and CIFAR-100, while ImageNet16-120 shows a slightly weaker correlation. This suggests that SEval-NAS effectively predicts hardware costs in the TSS space due to the architectural features. The reason why the SEval-NAS effectively predicts hardware costs in the TSS in Fig. 2 is due to the Autograd Traversal and String Generation block, which significantly optimizes the neural architecture in topology.



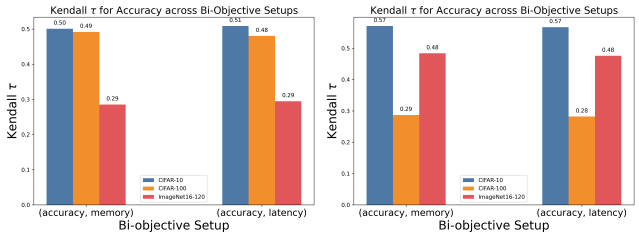

(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet16-120

(d) CIFAR-10 (e) CIFAR-100 (f) ImageNet16-120

**Figure 3: Plots of predicted vs true hardware cost of NATS-Bench SSS architectures (using T5-small) for performance metrics reported on CIFAR-10, CIFAR-100, and ImageNet16-120 datasets. The strength of correlation increases as $\tau$ approaches 1.**

For the NATS-Bench SSS results illustrated in Fig. 3, predicted memory usage again shows a strong positive correlation with the true values across all datasets. Predicted latency has a similar trend with a strong correlation in CIFAR-10 and CIFAR-100. However, latency prediction for the ImageNet16-120 dataset appears less robust, with noticeable variance. In terms of predicted latency on SSS across CIFAR-10, CIFAR-100, and ImageNet16-120 in Fig. 2. From the dataset itself, especially ImageNet16-120 compared to CIFAR-100 and CIFAR-10, we know ImageNet16-200 emphasizes low-resolution feature extraction; thus, ImageNet16-120 is reliable for memory predictions but unstable for latency prediction. Whereas CIFAR-100 and CIFAR-10 are designed for fine-grained classification, which results in good reliability on both memory predictions and latency memory. Therefore, while SEval-NAS remains reliable for memory predictions, its reliability in predicting latency is dataset-dependent and influenced by the variability of the architectures.

In comparison, while positively correlated, accuracy predictions in Fig. 4 exhibit weaker correlations than those for hardware costs. This suggests that SEval-NAS struggles to confidently infer accuracy from neural architecture representations. Furthermore, there is no trend linking dataset type to prediction reliability for accuracy, highlighting that accuracy depends on factors beyond straightforward architectural features. Overall, SEval-NAS demonstrates stronger predictability for hardware metrics than accuracy, primarily because hardware costs are directly tied to architectural characteristics. For example, more convolutional filters in a network




**(a) NATS-Bench TSS** **(b) NATS-Bench SSS**

**Figure 4: Kendall's $\tau$ correlation for predicted vs true accuracy in NATS-Bench TSS and SSS search spaces. Comparison includes both *(accuracy, latency)* and *(accuracy, memory)* bi-objectives.**

will need more computation than fewer convolutional filters [22]. Meanwhile, this correlation cannot be directly made for accuracy.

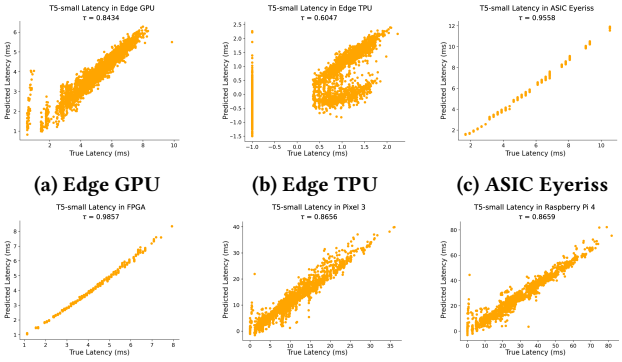
We also conducted an experiment (Appendix A) to evaluate three different encoder/decoder models (T5-small, T5-base, and T5-large) on NATS-Bench SSS and NATS-Bench TSS. This ablation study compared how model size affects the correlation between predicted versus true memory and predicted versus true latency, as measured by Kendall correlation.

The results from our extra ablation studies (Fig. 6 in Appendix A.1) showed that T5-small, T5-base, and T5-large perform similarly on the TSS benchmark. However, we observed that only T5-large encoders exhibit lower Kendall $\tau$ correlations on the SSS. Additionally, we found that different encoder sizes (Appendix A.2) do not significantly impact performance on NATS-Bench TSS for either memory or latency correlations (Fig. 7). In contrast, T5-large demonstrates weaker $\tau$ correlations for both memory and latency predictions on NATS-Bench SSS, as shown in Fig. 8.

## 4.4 Experiment 2: Predicting Hardware Cost (Evaluation on HW-NAS-Bench)

Although we observe a strong positive latency correlation in the NATS-Bench search spaces, the latency of a NN largely depends on the hardware environment. To investigate how well SEval-NAS would theoretically predict latency across various hardware devices, we evaluate its performance on HW-NAS-Bench.

HW-NAS-Bench [15] was designed for hardware-aware NAS. It includes two NAS search space designs: NAS-Bench-201's cell-based search space and FBNet's search space. The dataset provides the hardware cost of the NNs from both search spaces on commercial devices, including Edge GPU, Edge TPU, ASIC Eyeriss, FPGA, Pixel 3, and Raspberry Pi 4. FBNet search space [30] builds a layer-wise search space with a fixed macro-architecture and varying middle layers that can be searched. The architectures in this search space have regular structures that include nine cell candidates and 22 positions, yielding $9^{22} \approx 10^{21}$ different architectures. Due to the excessively large size of the search space, we do not use it in our experiment. NAS-Bench-201 search space [8] is the original search space of the TSS architectures in NATS-Bench. It contains the same 15,625 architectures with results reported for CIFAR-10, CIFAR-100, and ImageNet16-120 and their hardware costs on each of the six devices. We evaluate SEval-NAS on the HW-NAS-Bench's NAS-Bench-201 subspace for values reported on the CIFAR-10 dataset. The evaluator is trained to predict only latency, testing the performance of SEval-NAS on a single objective metric.




**(a) Edge GPU** **(b) Edge TPU** **(c) ASIC Eyeriss**

**(d) FPGA** **(e) Pixel 3** **(f) Raspberry Pi 4**

**Figure 5: Plots of predicted vs true latency for NAS-Bench-201 architectures with T5-small encoder for 6 edge devices reported in the HW-NAS-Bench benchmark. The strength of correlation increases as $\tau$ approaches 1.**

Results in Fig. 5 show a strong positive Kendall $\tau$ correlation for most edge devices, with values ranging from 0.6047 to 0.9742, demonstrating SEval-NAS's ability to predict latency across six different edge devices. The Edge TPU's latency predictions stand out as an outlier, showing a weaker correlation ($\tau = 0.6047$). This is attributed to negative latency values reported in the HW-NAS-Bench dataset, which likely affected the model's ability to predict latency in this case. Despite this, other devices, such as the Edge GPU ($\tau = 0.8676$), Eyeriss ($\tau = 0.9558$), FPGA ($\tau = 0.9742$), Pixel 3 ($\tau = 0.8599$), and Raspi4 ($\tau = 0.8659$), exhibit strong correlations, indicating consistent and reliable performance across diverse hardware configurations.

Both experiments show that training the evaluator for bi-objectives in NATS-Bench and a single objective in HW-NAS-Bench consistently yielded positive Kendall $\tau$ correlation values. This demonstrates the effectiveness of SEval-NAS in adapting to different numbers of evaluation objectives, further solidifying its strength as a predictive method for latency across edge devices.

We also conduct the additional experiments (in Appendix A.3) to run three different encoder/decoder models: T5-small (Fig. 5), T5-base (Fig. 9), and T5-large (Fig. 10) on HW-NAS-Bench as an ablation study of comparison of model size on the predicted latency vs true latency.

We observe that different encoder/decoder models do not have great impact (less than 0.02 latency correlation difference) except that T5-base in Edge GPU (0.8804) and T5-large in Edge GPU (0.8852) have stronger latency correlation than T5-small in Edge GPU(0.8434) since T5-small's operators are smaller, which results in GPU kernel launching overhead accounting for a higher proportion of the total latency. Moreover, T5-large and T5-base have longer latency, the relative impact of noise is smaller, which leads to a higher latency correlation.

**Table 1: Test accuracy and time for various NAS algorithms for NATS-Bench.**

| | CIFAR 10 | | CIFAR 100 | | ImageNet16-120 | |
|---|---|---|---|---|---|---|
| Algorithm | Accuracy | Time (s) | Accuracy | Time (s) | Accuracy | Time (s) |
| NASWOT (1000) | 93.10 ± 0.31 | 248 | 69.10 ± 1.61 | 248 | 45.08 ± 1.55 | 248 |
| TENAS | 93.90 ± 0.47 | 1558 | 71.24 ± 0.56 | 1558 | 42.38 ± 0.46 | 1558 |
| NASI | 93.55 ± 0.10 | 120 | 71.20 ± 0.14 | 120 | 44.84 ± 1.41 | 120 |
| GA-NINASWOT | 93.70 ± 0.63 | 206 | 71.57 ± 1.37 | 206 | 45.18 ± 2.05 | 206 |
| EPE-NAS | 91.31 ± 1.69 | 104 | 69.58 ± 0.83 | 104 | 41.84 ± 2.06 | 104 |
| FreeREA | 94.36 ± 0.00 | 45 | 73.51 ± 0.05 | 45 | 46.34 ± 0.00 | 45 |
| FreeREA + Latency | 94.36 ± 0.00 | 77 | 73.51 ± 0.00 | 82 | 46.34 ± 0.00 | 81 |
| FreeREA + Memory | 84.21 ± 14.91 | 34 | 49.67 ± 11.35 | 37 | 19.66 ± 7.19 | 38 |

## 4.5 Experiment 3: Ease of Integration (Evaluator in a NAS)

The contrast in correlation values observed between accuracy (low and moderate correlation) and hardware costs (strong correlation) highlights the evaluator's strength as a hardware cost predictor. This strength can be effectively leveraged to enhance NAS approaches that traditionally optimize accuracy as a single objective by incorporating hardware constraints into their search strategies. To demonstrate this applicability, we integrate SEval-NAS into FreeREA [2] to search in the NATS-Bench search space and define hardware constraints for the search. FreeREA uses evolutionary search to find candidate architectures and evaluates the candidate architecture using a training-free metric to estimate the accuracy performance.

FreeREA algorithm on NATS-Bench TSS is constrained by *FLOPS* and *#Params*. Since FLOPS is a poor proxy for hardware costs [15, 29], we replace it with two alternative constraints: 1) latency and 2) memory usage, each tested separately. For each metric, the mean value reported in the benchmark is set as the threshold for ranking candidate architectures. For example, these cases use the mean latency (45.96 seconds) and memory usage (166.67 MB) from the NATS-Bench CIFAR-10 dataset as thresholds. The resulting performance is compared against FreeREA and other training-free NAS algorithms reported in [2], with findings summarized in Table 1.

Latency-constrained search identified an average of approximately 230 architectures satisfying the threshold, achieving final average accuracies consistent with those reported in the original FreeREA study. In contrast, the memory-constrained search discovered fewer architectures (on average, fewer than 10) that met the threshold, indicating a bias in the search algorithm against low-memory architectures. This smaller pool of candidates led to higher variability in test accuracies due to the diverse performance of low-memory architectures. Importantly, while the latency-constrained search doubled FreeREA's search time, this overhead from evaluator inference remained negligible compared to other NAS algorithms. Conversely, the memory-constrained search required less time due to the limited number of viable candidate architectures.

These results highlight the flexibility of SEval-NAS for hardware cost evaluation while making minimal changes to the algorithm. By integrating additional constraints (e.g., latency and memory thresholds), the NAS algorithm can be tailored to select candidate architectures suitable for target hardware devices. For instance, deploying SEval-NAS on an edge device with memory constraints matching the device's operating range would yield architectures suitable for deployment. This study establishes the feasibility of incorporating SEval-NAS into NAS pipelines, with the specific objective of demonstrating integration viability rather than algorithmic optimization. The investigation of threshold parameter effects on search dynamics represents a natural extension of this foundational work and constitutes a promising direction for future research.

## 5 Conclusion and Future Work

NAS discovers novel architectures without expert knowledge, but suffers from extensive evaluation times when training or deploying architectures. We proposed SEval-NAS, which converts neural architectures to string representations via autograd graph traversal, then maps embeddings to predicted performance metrics. We evaluated SEval-NAS using two NAS benchmarks: NATS-Bench and HW-NATS-Bench, focusing on accuracy, latency, and memory. Our experimental results demonstrated that latency and memory predictions correlate best, indicating SEval-NAS's strength as a hardware cost predictor. However, its accuracy predictions showed moderate correlation, reflecting limitations in its ability to evaluate accuracy effectively. Our ablation studies on different sizes of encoder/decoder models on NATS-Bench and HW-NATS-Bench found that the larger the encoder, the lower the Kendall $\tau$ correlations on the NATS-Bench SSS. In terms of Hardware testing on HW-NATS-Bench, the experiment indicates a larger encoder/decoder model, stronger latency correlation in Edge GPU due to kernel operator, and longer latency. To test the adaptability of SEval-NAS, we incorporated it into FreeREA [2], adding latency and memory constraints to the search. The results showed that SEval-NAS had a low impact on search time and facilitated adding new evaluation criteria for selecting candidate architectures with minimal changes to the algorithm. These findings also showed that SEval-NAS can complement training-free NAS focused on predicting accuracy, providing a comprehensive evaluation of candidate architectures for diverse performance metrics.

Our experiments relied on hardware metrics reported in the respective NAS benchmarks, which may not accurately reflect the actual values if run on the devices. This shortcoming can be addressed by designing an on-device NAS with a lightweight SEval-NAS to evaluate candidate architectures. This enhancement and exploring additional thresholds in FreeREA are left as a direction for future work.

## Acknowledgement

This work has been supported by NSERC Discovery Grant No RGPIN 2025-00129.

## References

[1] Bowen Baker et al. 2018. Accelerating neural architecture search using performance prediction. In *International Conference on Learning Representations*.

[2] Niccoì O Cavagnero et al. 2023. Freerea: training-free evolution-based architecture search. In *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 1493–1502.

[3] Angelica Chen et al. 2023. Evoprompting: language models for code-level neural architecture search. *Advances in Neural Information Processing Systems*, 36, (Dec. 2023).

[4] Wuyang Chen et al. 2021. Neural architecture search on imagenet in four gpu hours: a theoretically inspired perspective. In *International Conference on Learning Representations*.

[5] Krishna Teja Chitty-Venkata et al. 2023. Neural architecture search benchmarks: insights and survey. *IEEE Access*, 11, 25217–25236.

[6] Patryk Chrabaszcz et al. 2017. A downsampled variant of imagenet as an alternative to the cifar datasets. (2017). arXiv: 1707.08819 [cs.CV].

[7] Xuanyi Dong et al. 2022. Nats-bench: benchmarking nas algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44, (July 2022), 3634–3646, 7, (July 2022).

[8] Xuanyi Dong and Yi Yang. 2020. Nas-bench-201: extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*.

[9] Yanjie Gao et al. 2020. Estimating gpu memory consumption of deep learning models. *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 20, (Nov. 2020), 1342–1352. ISBN: 9781450370431. doi:10.1145/3368089.3417050.

[10] Mohamed Imed Eddine Ghebriout et al. 2024. Harmonic-nas: hardware-aware multimodal neural architecture search on resource-constrained devices. In *Asian Conference on Machine Learning*. PMLR, 374–389.

[11] Amir Gholami et al. 2018. Squeezenext: hardware-aware neural network design. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 1638–1647.

[12] Minghao Guo et al. 2019. Irlas: inverse reinforcement learning for architecture search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[13] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images.

[14] Achintya Kundu et al. 2023. Transfer-once-for-all: ai model optimization for edge. *IEEE International Conference on Edge Computing and Communications (EDGE)*. ISBN: 9798350304831. doi:10.1109/EDGE60047.2023.00017.

[15] Chaojian Li et al. 2021. Hw-nas-bench:hardware-aware neural architecture search benchmark. *ICLR 2021 - 9th International Conference on Learning Representations*, (Mar. 2021).

[16] Yuke Li et al. 2023. Pareto optimization of cnn models via hardware-aware neural architecture search for drainage crossing classification on resource-limited devices. *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, (Nov. 2023), 1767–1775. ISBN: 9798400707858. doi:10.1145/3624062.3624258.

[17] Chenxi Liu et al. 2018. Progressive neural architecture search. In *European Conference on Computer Vision (ECCV)*.

[18] Renqian Luo et al. 2021. Neural architecture optimization. In *Neural Information Processing Systems*. https://github.com/renqianluo/NAO..

[19] Xiangzhong Luo et al. 2020. Edgenas: discovering efficient neural architectures for edge systems. *Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 2020-October, (Oct. 2020), 288–295.

[20] Bo Lyu et al. 2022. Resource-constrained neural architecture search on edge devices. *IEEE Transactions on Network Science and Engineering*, 9, 1.

[21] Joe Mellor et al. 2021. Neural architecture search without training. In *International conference on machine learning*. PMLR, 7588–7598.

[22] Atah Nuh Mih et al. 2024. Achieving Pareto Optimality using Efficient Parameter Reduction for DNNs in Resource-Constrained Edge Environment. *Proceedings of the Canadian Conference on Artificial Intelligence*, (May 2024). https://caiac.pubpub.org/pub/2gh9r4xc.

[23] Hieu Pham et al. 2018. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*. PMLR.

[24] Colin Raffel et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21, 140, 1–67.

[25] Esteban Real et al. 2019. Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*.

[26] Pengzhen Ren et al. 2021. A comprehensive survey of neural architecture search: challenges and solutions. *ACM Computing Surveys (CSUR)*, 54, 4, 1–34.

[27] Blake Richey et al. 2024. Multi-reward optimization using genetic algorithms for edge ai. In *Real-Time Image Processing and Deep Learning 2024*. SPIE.

[28] Matteo Risso et al. 2022. Lightweight neural architecture search for temporal convolutional networks at the edge. doi:10.1109/TC.2022.3177955.

[29] Nilotpal Sinha et al. 2024. Hardware aware evolutionary neural architecture search using representation similarity metric. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2628–2637.

[30] Bichen Wu et al. 2019. Fbnet: hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE/CVF conference on computer vision and pattern recognition*.

[31] Lingxi Xie and Alan Yuille. 2017. Genetic cnn. In *International Conference on Computer Vision*. IEEE.

[32] Li Lyna Zhang et al. 2021. Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *International Conference on Mobile Systems, Applications, and Services*. ACM.

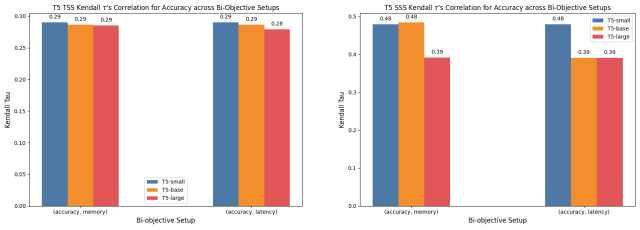
[33] Yusen Zhang et al. 2024. Oncenas: discovering efficient on-device inference neural networks for edge devices. *Information Sciences*.

[34] Barret Zoph et al. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8697–8710.

[35] Barret Zoph and Quoc Le. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.


# A Appendix

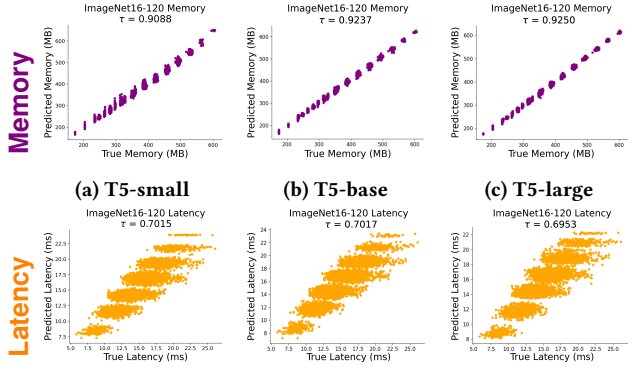
## A.1 Ablation Study 1: Evaluation among different T5 encoders across Bi-Objective Setups)




(a) NATS-Bench TSS (b) NATS-Bench SSS

**Figure 6: Kendall's $\tau$ correlation for predicted vs true accuracy among encoders of T5-small, T5-base, and T5-large on NATS-Bench TSS and NATS-Bench SSS *(accuracy, latency)* and *(accuracy, memory)* bi-objectives.**

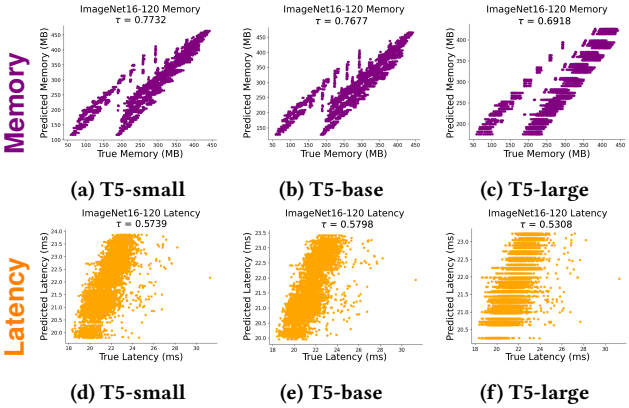
## A.2 Ablation Study 2: Feasibility Testing (Evaluation on NATS-Bench) on Different Encoders




(a) T5-small (b) T5-base (c) T5-large

(d) T5-small (e) T5-base (f) T5-large

**Figure 7: Plots of predicted vs true hardware cost of NATS-Bench TSS architectures for performance metrics reported on T5-small, T5-base, and T5-large. The strength of correlation increases as $\tau$ approaches 1.**

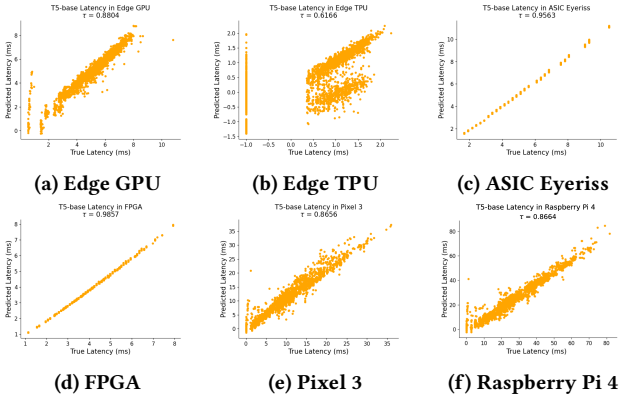



(a) T5-small (b) T5-base (c) T5-large

(d) T5-small (e) T5-base (f) T5-large

**Figure 8: Plots of predicted vs true hardware cost of NATS-Bench SSS architectures for performance metrics reported on T5-small, T5-base, and T5-large. The strength of correlation increases as $\tau$ approaches 1.**

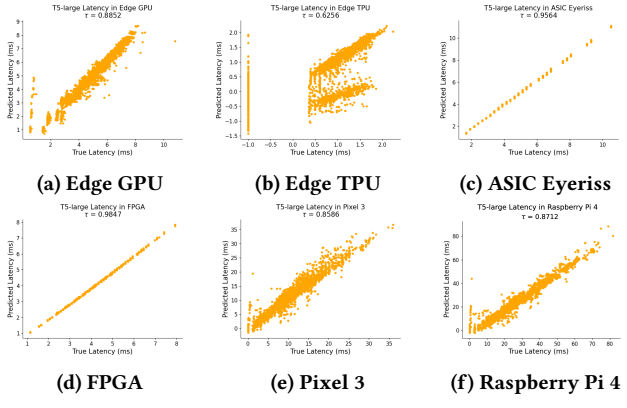
## A.3 Ablation Study 3: Predicting Hardware Cost (Evaluation on HW-NAS-Bench) on Different Encoders




(a) Edge GPU (b) Edge TPU (c) ASIC Eyeriss

(d) FPGA (e) Pixel 3 (f) Raspberry Pi 4

**Figure 9: Plots of predicted vs true latency for NAS-Bench-201 architectures with T5-base encoder for 6 edge devices reported in the HW-NAS-Bench benchmark. The strength of correlation increases as $\tau$ approaches 1.**




(a) Edge GPU (b) Edge TPU (c) ASIC Eyeriss

(d) FPGA (e) Pixel 3 (f) Raspberry Pi 4

**Figure 10: Plots of predicted vs true latency for NAS-Bench-201 architectures with T5-large encoder for 6 edge devices reported in the HW-NAS-Bench benchmark. The strength of correlation increases as $\tau$ approaches 1.**