# Variation Matters: from Mitigating to Embracing Zero-Shot NAS Ranking Function Variation

**Anonymous authors**
**Paper under double-blind review**

## Abstract

Neural Architecture Search (NAS) is a powerful automatic alternative to manual design of a neural network. In the zero-shot version, we use fast ranking functions to compare architectures without training them. The outputs of the ranking functions often vary significantly due to different sources of randomness, including the evaluated architecture's weights' initialization or the batch of data used for calculations. A common approach to addressing the variation is to average a ranking function output over several evaluations. We propose taking into account the variation in a different manner, by viewing the ranking function output as a random variable representing a proxy performance metric. During the search process, we strive to construct a stochastic ordering of the performance metrics to determine the best architecture. Our experiments show that the proposed stochastic ordering can effectively boost performance of a search on standard benchmark search spaces.

## 1 Introduction

Zero-shot Neural Architecture Search (NAS) is a way to perform selection of a neural architecture without training every candidate architecture. Recently, zero-shot NAS has been applied to a variety of practical tasks, including medical image processing (Wang et al., 2024), sequence processing (Serianni & Kalita, 2023), language model architecture enhancements (Javaheripi et al., 2022) and general image classification (Jiang et al., 2024; Li et al., 2022).

In general, a zero-shot approach consists of two parts: a ranking function and a search algorithm (Xu et al., 2021; Rumiantsev & Coates, 2023; Cavagnero et al., 2023a; Li et al., 2022; Jiang et al., 2024). The ranking function aims to capture the quality of a particular candidate architecture in the form of a scalar value, which allows two architectures to be easily compared (Abdelfattah et al., 2021). A search algorithm uses a ranking function to determine the best architecture over the given *architectural search space*. For this work, we view a search space as a combination of a feasible architecture set and a dataset of labelled training, validation, and test samples.

While the zero-shot approach is the fastest way to find an architecture that performs well, it suffers from variations in the output of the ranking function. In order to assess an architecture as quickly as possible, most zero-shot techniques sample a single batch of data, and randomly initialize the architecture. The ranking function then uses the batch of data and the initialized network to derive a scalar score. The batch selection and initialization thus induce considerable randomness in the output. Although there are several data-agnostic ranking functions, including SynFlow (Abdelfattah et al., 2021), LogSynFlow (Cavagnero et al., 2023a), and Zen-Score (Lin et al., 2021), these use a batch of *randomly generated* data as an input. Thus, there is still randomness, and the same problem remains. Typically, zero-shot techniques address this issue by averaging the outputs of the ranking function over multiple batches of data.

A zero-shot search procedure can counter some of the variation of the preferred ranking function by employing a robust search algorithm. For example, Chen et al. (2021) use prune-based search to increase the stability of the search. In general, the performance of a search algorithm can vary considerably, depending on which ranking function is paired with it (Akhauri et al., 2022; Rumiantsev & Coates, 2023). Since it is very challenging to meaningfully analyse a ranking function in isolation (independently of a search algorithm),

we base our experimental conclusions on pairs of ranking function and search algorithm. We focus on the most popular ranking functions and search procedures.

In this paper, we assess the variation of different ranking functions, explore how common ways to minimize the ranking function variation help to increase the zero-shot performance, and introduce a way to incorporate variation into the assessment of an architecture. Our main contributions are:

- We introduce a ranking function variance analysis framework for zero-shot NAS researchers to analyse existing and newly designed ranking functions.
- We use the variance of a ranking function to statistically compare architectures, and show that this enhances the performance of existing zero-shot architecture search algorithms.

## 2 Background

### 2.1 Search Algorithms

There are three common choices of search algorithms that are used in Zero-Shot NAS: random search, prune-based search and evolutionary search. **Random search** is the fastest search algorithm. It randomly samples a subset of architectures from the search space, and then selects the one with the highest rank according to the chosen ranking function. The size of the sampled subset is the main hyperparameter that controls the trade-off between search speed and efficacy. Due to its simplicity, it is a common choice (Lopes et al., 2021; Xu et al., 2021; Mellor et al., 2021; Mok et al., 2022). Random search suffers the most from variability in the ranking function outputs.

**Evolutionary search** procedures are based on genetic algorithms. They start with randomly selected architectures, then generate mutations, and then select a subset that are ranked the highest according to the ranking function output. The procedure is then iterated. Mutations can be based on replacement of architectural operators or involve direct manipulations of the architecture encoding. Evolutionary search is slower and more challenging to implement than random search, but the results it produces are more stable. Variations of this search technique are employed more often in recent works (Lin et al., 2021; Akhauri et al., 2022; Li et al., 2022; Mok et al., 2022; Cavagnero et al., 2023a;b; Yang et al., 2023; Wang et al., 2024). Often, the ranking function is applied to multiple batches of data, with the results averaged, in order to provide a more stable score. This procedure is usually performed once, when the architecture is first encountered during the search, with the calculated score cached for later use.

The third type of search algorithm is **prune-based search**. It iteratively removes edges from the hypergraph that can be used to represent all architectures. By estimating the difference in ranking function output for the architectures represented by the hypergraph with and without a given edge, the algorithm can choose the best edge to prune. This process is repeated until the hypergraph represents a single architecture. This type of search is the most stable and suffers the least from the variations in the ranking function. However, it is the slowest of all, the most challenging to implement, and requires that the architectural search space is specified as a hypergraph. As a result, prune-based search is rarely adopted (Chen et al., 2021; Rumiantsev & Coates, 2023; Jiang et al., 2024).

### 2.2 Zero-shot ranking functions

Many zero-shot ranking functions have been proposed, and we now review some of the most prominent. Abdelfattah et al. (2021) repurposed metrics previously employed in neural network pruning. The saliency metrics `Snip` (Lee et al., 2019), `Grasp` (Wang et al., 2020), `Fisher` (Turner et al., 2020) and `SynFlow` (Tanaka et al., 2020) measure the impact of removing a specific parameter or operation from the network. In adapting these to construct a zero-shot ranking function, Abdelfattah et al. (2021) sum over all of the architectural parameters, and thus construct a measure of the fitness of the entire architecture. Cavagnero et al. (2023a) introduced a logarithmic version of `SynFlow` named `LogSynFlow`. Applying a logarithm suppresses some of the gradient explosions that can unduly influence the score.

Mellor et al. (2021) proposed the first two ranking functions designed specifically for the zero-shot setting: the `Eigenvalue score` and the Hamming distance between ReLU regions. Based on the Neural Tangent Kernel (NTK) (Jacot et al., 2018), the `Eigenvalue score` uses the eigenvalues of the gradient correlation matrix to predict an architecture's learning capabilities. An alternative measure is the number of activated ReLU regions for a sampled batch of data Xiong et al. (2020). The underlying conjecture is that activation of more regions allows the network to learn more complex data dependencies. The `Entropic Score` (Cavagnero et al., 2023b) is also based on ReLU activations.

Several other ranking functions are based on the NTK, including (i) the mean value of the NTK (Xu et al., 2021); (ii) the Frobenius norm of the NTK (Xu et al., 2021); and (iii) the negative condition number of the NTK (Chen et al., 2021). The NTK can introduce undesirable computational overhead, so other researchers have explored alternative metrics. Rumiantsev & Coates (2023) show that the Frobenius norm of the Neural Network Gaussian Process (NNGP) kernel can act as an effective ranking function. Jiang et al. (2024) focus on the Pearson correlation kernels of the internal representations of an architecture, and compute the sum of the smallest eigenvalues of these kernels. For all of these proposed ranking functions, the required kernels are estimated using a random batch of data.

Since most benchmark search spaces focus on the classification task, some ranking functions incorporate the available label information. Label-Gradient Alignment (`LGA`) (Mok et al., 2022) constructs a metric based on both the class similarity matrix and the NTK. Lopes et al. (2021) propose `EPE-NAS`, which uses the gradient correlation matrix, and is thus similar to the `Eigenvalue score`, but is computed class-wise. For practical reasons, both `LGA` and `EPE-NAS` are computed using a random batch of data.

Some ranking functions are based on the gradients of the trainable weights. `SweetGrad` counts the number of gradient norms in a predefined interval (Yang et al., 2023). Li et al. (2022) sum the coefficients of variations associated with the weight gradients to compute `ZiCo`.

## 3 Problem statement

A neural architecture search space can be represented as a graph $G_X$, where each node corresponds to an architecture and an edge represents a single architectural change. Associated with the architecture space is a dataset $\mathcal{D}$ that consists of pairs of features $x \in \mathcal{X}$ and labels $y \in \mathcal{Y}$. In this work we consider computer vision tasks where an architecture $arch_i$ is associated with a final accuracy after training, labelled $acc_i$. A zero-shot ranking function assigns a scalar score to each architecture: $s_i = r(arch_i, \mathcal{D})$. A search is then conducted using these scalar scores, with the goal of selecting the most accurate architecture.

As discussed above, almost every zero-shot ranking function has an element that is evaluated over a batch of input data (e.g., the NTK, gradient coefficients of variations, or logit similarity matrix). To improve the stability of the output, most ranking functions are averaged over multiple random batches. Our goal is to use the multiple ranking function outputs in a more effective way.

In subsequent sections, we explore how sensitive different ranking functions are to the random selection of the input data used in their calculation. We point out how commonly used statistical methods allow us to overcome such sensitivity in order to improve the performance of a zero-shot approach.

## 4 Methodology

This section describes our study into the variability of ranking functions. First, we introduce a variation metric for assessing the variability of a ranking function for a particular search space and batch size. Second, we present our zero-shot search setup and explain how the variation metric can be used to improve performance.

### 4.1 Ranking function variation

It is common for ranking functions to use a real training dataset. In order to assign a score to an architecture a batch of data is randomly drawn from the dataset. The ranking function scores vary from batch to batch, with the extent of the variation being dependent on the batch size, the data set, and the task.

We measure ranking function variation over the entire search space by computing, for each architecture, an empirical coefficient of variation for the ranking function output over randomly sampled batches of input data. We then average over the architectures. Denoting the batch size by $B$ and the number of sampled batches by $K$, and recalling that there are $N$ architectures in the search space, we define the ranking function variation as:

$$Var_{SS}(r, B, K) \triangleq \frac{1}{N} \sum_{i=1}^{N} CV\left(\mathcal{M}_i(B, K)\right) \quad \text{for} \quad \mathcal{M}_i(B, K) = \{r(arch_i, d_k)\}_{k=1}^{K}, \tag{1}$$

where $r$ is a ranking function, $d_j$ is a batch of data of size $B$, uniformly sampled from the dataset associated with the search space $SS$, and $CV(\mathcal{X}) = \frac{Var(\mathcal{X})}{Mean(\mathcal{X})}$. A higher average coefficient of variation indicates greater variability with respect to the selection of input data.

Typically, if we know that a ranking function is highly variable with respect to the input data, then this motivates us to enhance the robustness by averaging over multiple random batches. On the other hand, if a ranking function exhibits less variability, then a score can be taken from a reduced number of randomly drawn batches. This leads to a faster search, because ranking function evaluation tends to be the most computationally expensive element of zero-shot NAS.

### 4.2 Search algorithm

Zero-shot architecture search uses a ranking function as a proxy to evaluate the quality of an architecture. In order to compare two architectures, a selected ranking function is evaluated on both of them and the outputs of the ranking function are compared. Assuming positive correlation between accuracy and the adopted metric, the architecture with the higher corresponding metric is considered to be better. The ranking function outputs are subject to randomness, due, for example, to random architecture initialization or random selected input batch. A common technique to mitigate randomness is averaging: a ranking function is evaluated multiple times on the same architecture and the output is averaged over the evaluations.

Instead of averaging, the key component of our approach is a statistical comparison of the architectures. We treat multiple evaluations of the ranking metric as multiple observations of a random variable. We then test for *stochastic dominance*. Viewing the ranking function metrics for two compared architectures as random variables, $X$ and $Y$, we test

**Algorithm 1** Statistical MAX and TOP-K pseudocode

**Ensure:** Each $r_i$ is a list of $N$ ranking function measurements

```
function STAT-MAX(r_1, ..., r_T)
    max ← r_1
    for i ← 2, T do
        /* Mann-Whitney U-test */
        p ← MannWhitneyU(max, r_i)
        /* Threshold is a significance level */
        if p < Threshold then
            max ← r_i
        end if
    end for
    return max
end function

function STAT-TOPK(k, r_1, ..., r_T)
    top ← ∅
    rs ← r_1, ..., r_T
    for i ← 1, k do
        m ← STAT-MAX(rs_1, rs_2, ...)
        top ← top ∪ {m}
        rs ← rs \ {m}
    end for
    return top
end function
```

whether $p(X > k) \geq p(Y > k)$ for all $k$
(and $p(X > k) > p(Y > k)$ for some $k$). This test differs from comparing the mean performance, $\bar{X} > \bar{Y}$. Stochastic dominance implies $\bar{X} > \bar{Y}$, but the converse is not true. Our testing process thus targets a stricter performance discrepancy. This makes the search more effective and efficient, because we navigate towards architectures that are more likely to be genuinely superior.

To perform the comparison, we use the U-value from the Mann-Whitney U-test (Mann & Whitney, 1947). This statistical test assesses the null hypothesis that both sets of observations are drawn from the same distribution. The alternative hypothesis (for a one-sided test) is that the random variable X is stochastically greater than the random variable Y. By comparing the Mann-Whitney U values for the two samples, we can identify one as a superior architecture. We declare such superiority only if the null hypothesis can be rejected at a predetermined significance level (e.g., 0.05). If this is not the case, we cannot meaningfully differentiate between the architectures in terms of predicted performance.

We use two popular search algorithms in our work: *random* search and *evolutionary* search. Both algorithms are easy to implement for an arbitrary search space and require few additional considerations when paired with a ranking function in a zero-shot setting. For **random search** (Mellor et al., 2021) we sample $N$ architectures randomly from the search space. Then we evaluate a ranking function on each sampled architecture $M$ times. We use the Mann-Whitney U-test to rank the candidates. In the case of ties, we select an architecture randomly from the tied candidates.

The REA (Real et al., 2019) **evolutionary search** starts by sampling a *population* of $N$ architectures randomly from the search space. During each iteration, a subset of $n$ exemplars (architectures) is randomly sampled from the population. We order the exemplars using the Mann-Whitney U-test, with random tie-breaking. The exemplar that is first in this ordering is selected and mutated. In our experiments, we use an operator replacement mutation, so that the original exemplar and the mutated exemplar have an architectural edit distance of one. We evaluate a ranking function on the mutant $M$ times and include the mutated architecture in the population, removing the lowest ranked exemplar (with random tie-breaking). After $T$ iterations of this procedure, we use the Mann-Whitney U-test to order the population and select the highest ranked.

The FreeREA (Cavagnero et al., 2023a) evolutionary search procedure introduces a small modification to the REA algorithm. After a subset of $n$ exemplars is randomly sampled, two exemplars with the statistically highest rank are selected. Both exemplars are mutated and the two mutants are included in the population. *Crossover* is performed by splicing together parts of the two selected exemplars, and the resultant crossover exemplar is also included into the population to increase the exploration capabilities of the search process. We provide pseudocode for the evolutionary algorithms in Appendix B.

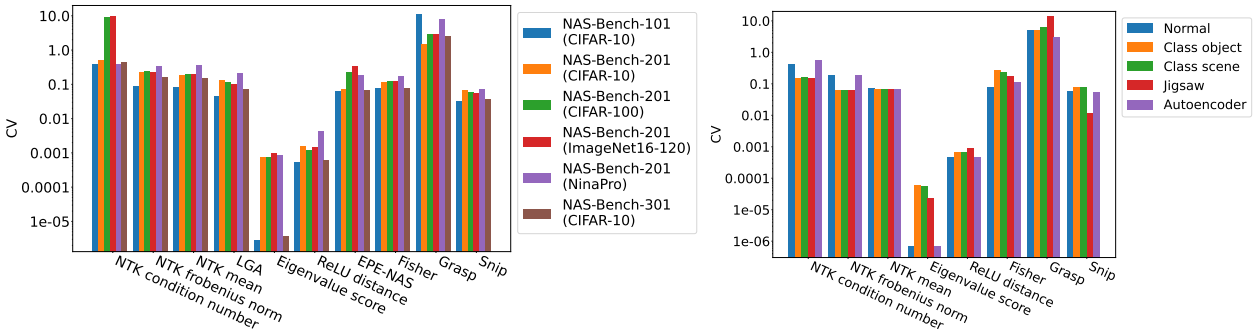
## 5 Experiments

### 5.1 Ranking function variation

In the first experiment, we show how different ranking functions are influenced by a randomly chosen batch of data and examine how consistent this behaviour is over a variety of search spaces. A lower coefficient of variation means that a ranking function is generally more reliable, can provide more consistent output, and requires fewer batches.

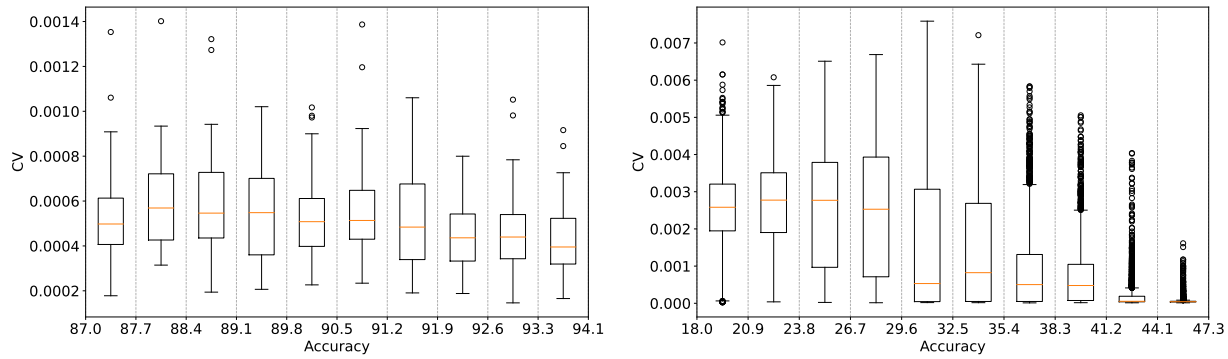
We measure the variation $Var_{SS}(r, B, K)$ by evaluating equation 1. Recall that $SS$ denotes the search space, $r$ denotes the ranking function, $B$ is the batch size, and $K$ is the number of evaluations. In this experiment, in order to focus on the most influential source of variability – the input batch – we initialize each network once, before the first batch is selected. We set $K = 10$ and set $B = 64$. An output of a ranking function is averaged over 10 runs for different random batches given the same initialisation for the network.

We present results for the NAS-Bench search spaces in Fig. 1a and results for the TransNAS search spaces in Fig. 1b. Overall, Eigenvalue score and ReLU Hamming distance are by far the most stable ranking functions. Snip is more stable than the others. Fisher exhibits stability similar to that of Snip on the NAS-Bench search spaces, which are considered easy, but performs worse on TransNAS-Bench. Fisher and NTK-based kernels




(a) (b)

Figure 1: Ranking functions variation ($Var_{SS}(r, B, K)$ for ranking function $r$, batch size $B = 64$, and search space $SS$) on (a) NAS-Bench and (b) TransNAS-Bench-101 demonstrate that different ranking functions have considerably different variations within the search space, but a specific ranking function's variation remains similar for the different search spaces.




(a) (b)

Figure 2: Box plot (versus accuracy) of coefficients of variation for individual architectures ($CV(\mathcal{M}_i(B,K)) = \frac{Var(\mathcal{M}_i(B,K)}{Mean(\mathcal{M}_i(B,K)}$ where $\mathcal{M}_i(B,K) = \{r(arch_i, d_k)\}_{k=1}^{K}$ is a set of $K$ ranking function outcomes for architecture $arch_i$ derived from random batches of data $d_k$, each of size $B = 64$. The depicted ranking functions are ReLU Hamming distance on NAS-Bench-101 (CIFAR-10) (a) and Eigenvalue Score on NAS-Bench-201 (ImageNet16-120) (b). For visualization purposes, we exclude ten percent of the architectures-those that exhibit lowest accuracy.

have a performance in major dependence of a dataset that a search space uses. We emphasize that low variation is a desirable criterion for the ranking function selection, but it does not imply high performance.

For all ranking functions, the coefficient of variation of an architecture tends to decrease as the average accuracy increases, although this effect is more pronounced for some ranking function and search space combinations. In general, this is a result of the ranking function output varying less for the higher quality architectures. Fig. 2 presents example results for ReLU Hamming distance on NAS-Bench-101 (CIFAR-10) and Eigenvalue Score on NAS-Bench-201 (ImageNet16-120). Due to the very high variability often exhibited by poorly performing architectures, we exclude the worst-performing decile from the analysis, and present box plots of the coefficients of variation (for individual architectures) versus accuracy on the test set. We observe a negative trend in each case, although it is much more evident for Eigenvalue Score.

The negative trend suggests that we could consider directly using lower coefficient of variation as an indicative variable during the search. To further investigate this possibility, we measured Kendall-$\tau$ correlation between ranking function variation and architecture accuracy (see Fig. 3). In many cases, there is a strong negative

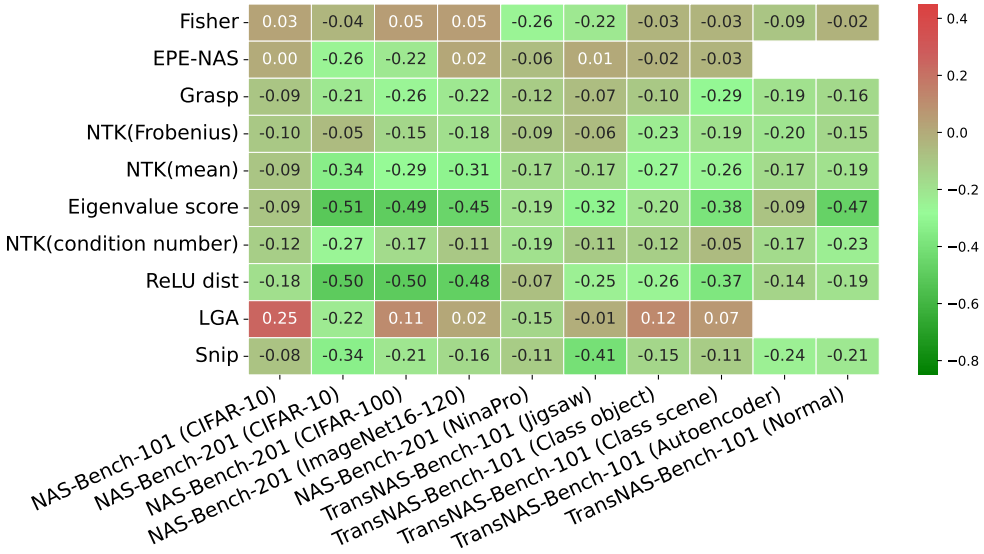



Figure 3: Kendall-$\tau$ correlation coefficient between a ranking function coefficient of variation values and validation accuracy, for each ranking function and search space. EPE-NAS and LGA require classification labels and therefore can not be computed for TransNAS-Bench-101 (Autoencoder) and TransNAS-Bench-101 (Normal).

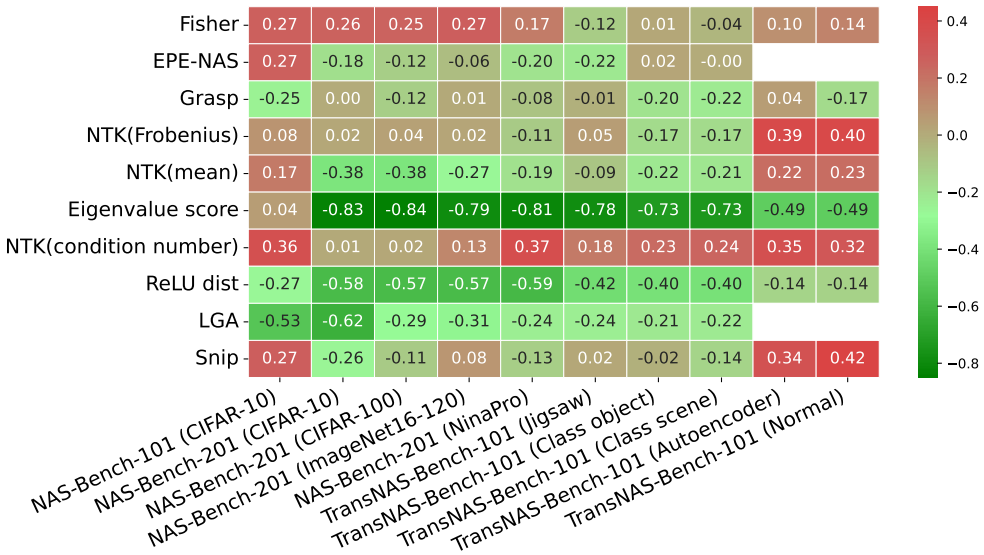



Figure 4: Kendall-$\tau$ correlation coefficient between a ranking function coefficent of variation values and ranking function mean value taken over 10 evaluations, for each ranking function and search space. EPE-NAS and LGA require classification labels and therefore can not be computed for TransNAS-Bench-101 (Autoencoder) and TransNAS-Bench-101 (Normal).

Table 1: Random search on $N$ sampled architectures with averaging (Avg) and statistic (Stat) evaluation. Mann-Whitney U-test is used for statistical evaluation. Each ranking function is evaluated 10 times for the architecture. We provide mean values and standard deviations over 100 runs.

| Search Space | Eigenvalue score | | ReLU dist | | NTK(cond) | |
|---|---|---|---|---|---|---|
| | Avg | Stat | Avg | Stat | Avg | Stat |
| **N**= 10 | | | | | | |
| NAS-Bench-101 (CIFAR-10) | 91.51±1.44 | 91.71±2.22 | 91.28±1.80 | 91.63±1.51 | 90.94±1.42 | 91.24±1.63 |
| NAS-Bench-201 (CIFAR-10) | 91.33±1.57 | 92.21±1.66 | 91.80±1.33 | 92.34±1.14 | 91.61±1.96 | 91.92±2.15 |
| NAS-Bench-201 (CIFAR-100) | 68.61±3.55 | 69.13±2.93 | 71.76±1.92 | 71.68±2.03 | 68.05±4.42 | 69.47±3.17 |
| NAS-Bench-201 (ImageNet16-120) | 40.59±5.05 | 41.22±4.56 | 40.69±4.09 | 41.24±3.33 | 40.91±5.45 | 41.64±4.76 |
| NAS-Bench-201 (NinaPro) | 90.13±1.30 | 90.40±1.40 | 90.34±1.83 | 90.76±1.89 | 90.77±1.35 | 90.99±1.38 |
| TransNAS-Bench-101 (Jigsaw) | 90.72±4.55 | 91.88±4.76 | 87.28±15.74 | 89.11±11.73 | 90.97±4.63 | 92.48±3.18 |
| TransNAS-Bench-101 (Class object) | 47.33±2.47 | 48.26±2.84 | 47.65±2.50 | 48.71±2.69 | 47.91±2.11 | 48.48±2.46 |
| TransNAS-Bench-101 (Class scene) | 59.94±5.30 | 60.52±5.14 | 60.87±5.09 | 61.70±4.11 | 60.90±3.53 | 61.58±1.73 |
| **N**= 100 | | | | | | |
| NAS-Bench-101 (CIFAR-10) | 91.71±1.32 | 92.06±1.15 | 91.98±2.08 | 91.98±2.01 | 89.67±2.25 | 90.06±1.98 |
| NAS-Bench-201 (CIFAR-10) | 91.10±1.61 | 91.73±1.43 | 92.11±0.98 | 92.52±1.10 | 91.09±1.84 | 91.75±2.38 |
| NAS-Bench-201 (CIFAR-100) | 68.43±2.82 | 69.05±2.64 | 70.38±1.77 | 70.75±1.79 | 69.39±4.50 | 70.01±4.36 |
| NAS-Bench-201 (ImageNet16-120) | 40.34±4.66 | 40.96±4.98 | 42.54±3.93 | 43.34±2.47 | 41.04±4.49 | 41.50±4.42 |
| NAS-Bench-201 (NinaPro) | 90.00±1.91 | 90.22±1.86 | 89.18±1.72 | 89.53±1.55 | 91.29±1.00 | 91.36±1.05 |
| TransNAS-Bench-101 (Jigsaw) | 89.95±5.57 | 90.59±4.10 | 90.58±3.55 | 85.33±2.21 | 92.00±2.78 | 91.99±2.96 |
| TransNAS-Bench-101 (Class object) | 47.04±4.06 | 49.46±4.56 | 48.25±3.11 | 48.87±2.96 | 49.48±0.97 | 49.81±1.77 |
| TransNAS-Bench-101 (Class scene) | 60.94±3.23 | 61.12±4.14 | 61.19±3.10 | 61.50±4.32 | 61.56±0.56 | 61.29±2.85 |
| **N**= 1000 | | | | | | |
| NAS-Bench-101 (CIFAR-10) | 91.47±1.36 | 91.57±1.31 | 92.12±2.45 | 92.60±1.73 | 90.54±2.35 | 90.92±2.40 |
| NAS-Bench-201 (CIFAR-10) | 91.25±1.46 | 92.08±1.64 | 92.04±1.22 | 92.69±1.08 | 91.00±2.07 | 91.88±2.16 |
| NAS-Bench-201 (CIFAR-100) | 69.32±3.17 | 70.63±2.68 | 70.78±1.66 | 71.52±1.14 | 69.83±2.62 | 70.40±2.58 |
| NAS-Bench-201 (ImageNet16-120) | 41.42±4.56 | 42.17±4.11 | 42.96±2.78 | 43.79±2.92 | 41.59±3.41 | 42.24±2.03 |
| NAS-Bench-201 (NinaPro) | 90.59±1.73 | 90.61±2.19 | 90.47±1.25 | 90.21±1.34 | 91.49±0.65 | 91.57±0.89 |
| TransNAS-Bench-101 (Jigsaw) | 88.63±4.32 | 90.81±3.93 | 90.60±3.00 | 88.52±11.35 | 92.40±0.98 | 92.57±1.51 |
| TransNAS-Bench-101 (Class object) | 44.64±3.26 | 45.46±4.21 | 44.99±3.31 | 45.82±5.97 | 44.98±1.15 | 45.36±0.89 |
| TransNAS-Bench-101 (Class scene) | 61.42±3.37 | 61.50±3.51 | 61.47±3.09 | 61.76±4.32 | 61.74±0.43 | 62.29±0.45 |

correlation, but there are exceptions. For example, there is very weak correlation between accuracy and the coefficient of variation of the Fisher metric. On some search spaces, LGA even exhibits a positive correlation, meaning that it is, on average, more unstable for the better-performing architectures. We conjecture that this behaviour arises due to the use of labels in both the LGA ranking function and the accuracy evaluation. EPE-NAS, the other ranking function that uses labels, exhibits close-to-zero (and occasionally slightly positive) correlation.

We can consider naive approaches to incorporate the coefficient of variation into the search process. For example, after appropriate normalization, we could sum a ranking function output with its coefficient of variation. Fig. 4 depicts the Kendall-$\tau$ correlation between the coefficient of variation of a ranking function and its mean value for different search spaces. For some ranking functions such as Eigenvalue score and ReLU distance, the magnitude of the correlation is high, meaning that the coefficient of variation does not provide much additional useful information for the search. By contrast, for other ranking functions such as Frobenius norm of the NTK and the condition number of NTK, the magnitude of the correlation is low. In experiments (see Appendix C) we observe that the search performance improvement of this naive scheme is indeed considerably greater for these ranking functions. Although a naive incorporation of the coefficient of variation can improve results of the search, its impact is inconsistent; the statistical comparison approach we propose in Section 4.2 exhibits more consistent improvement and is effective for all ranking functions.

### 5.2 Random search

In order to demonstrate our statistical approach, we first use a random search proposed by Mellor et al. (2021). Results are presented in Table 1. We selected Eigenvalue score, ReLU Hamming distance and condition number of NTK for this experiment. The first two serve as examples of ranking functions with low variation, while the final one exhibits high variation. To evaluate an architecture we randomly sample a batch of 64 and randomly initialise architecture weights. This evaluation simulates a typical case of

Table 2: Accuracy comparison between cached rank mapping (Cached) and computation on-the-fly (On-the-fly). Variation of ranking function influence on NAS-Bench-101/201. We provide mean values and standard deviations over 10 runs.

| Search Space | Search Alg. | Eigenvalue score | | ReLU dist | | NTK(cond) | |
|---|---|---|---|---|---|---|---|
| | | Cached | On-the-fly | Cached | On-the-fly | Cached | On-the-fly |
| NAS-Bench-201 (CIFAR-10) | Greedy | 92.76±0.90 | 92.30±0.90 | 92.36±0.31 | 91.91±3.85 | 88.28±1.30 | 86.85±0.26 |
| | REA | 93.02±0.32 | 91.98±4.11 | 93.15±0.15 | 90.74±4.24 | 86.07±2.56 | 85.06±3.13 |
| | FreeREA | 93.23±0.47 | 92.43±2.09 | 93.29±0.58 | 91.27±7.50 | 87.65±1.80 | 86.15±1.79 |
| NAS-Bench-201 (CIFAR-100) | Greedy | 69.54±1.01 | 65.42±1.37 | 70.00±0.03 | 65.83±2.11 | 63.32±1.89 | 64.18±4.28 |
| | REA | 71.54±0.80 | 67.42±1.37 | 71.83±0.61 | 68.02±1.50 | 68.75±4.71 | 65.61±5.19 |
| | FreeREA | 71.98±0.26 | 69.44±1.10 | 71.63±1.35 | 68.89±1.42 | 66.75±4.71 | 63.43±4.65 |
| NAS-Bench-201 (ImageNet16-120) | Greedy | 42.68±1.06 | 41.14±3.29 | 44.78±1.34 | 42.54±4.75 | 40.83±3.30 | 35.79±7.46 |
| | REA | 44.90±0.76 | 43.55±4.71 | 43.82±1.53 | 41.34±7.19 | 41.95±10.23 | 34.97±10.94 |
| | FreeREA | 44.84±3.17 | 40.62±5.28 | 43.82±1.23 | 42.58±5.24 | 39.51±8.03 | 36.60±10.34 |

ranking function usage. A ranking function is evaluated ten times on every queried architecture. In order to reduce the number of experiments we do not test combinations of ranking function. We provide results for averaged rank value and for the proposed statistical ranking approach. The 5% significance level is used for rejecting the null hypothesis when conducting the Mann-Whitney U-test. We repeat each experiment 100 times, computing the mean value of the accuracy for the selected architecture and its variation. Our findings demonstrate that our statistical approach is a better way to use the multiple outcomes than simple averaging. As we can see in Table 1, the proposed statistical method leads to improved performance for almost every ranking function and search space. The average accuracy improvement is 0.49. The improvement is small, but there is a negligible additional computational overhead. Our experiments do not expose a significant difference between the improvement for low- versus high-variance ranking functions. We observed that the largest search improvement is achieved on the NAS-Bench-201 search spaces, but that could be due to the fact that those search spaces are small. Less improvement is observed for the largest search space NAS-Bench-101 (CIFAR-10).

### 5.3 Evolutionary search

Before we examine the performance of the statistical comparison approach for evolutionary search, we demonstrate the importance of caching. As opposed to random search, where all architectures are sampled once, in evolutionary search, the same architecture can be encountered multiple times due to mutation or crossover. Due to the variations in the ranking function output, re-evaluation of a previously encountered architecture can lead to a performance degradation in the search. We demonstrate this in Table 2. We report mean values and standard deviations for accuracies over 10 runs. A randomly sampled batch of 64 with randomly initialise architecture weights are used to evaluate an architecture. In the case of caching, we evaluate the architecture only once, when it is first encountered, and then save the ranking function output. Subsequently, if the architecture is encountered again, the cached value of the ranking function is used in comparisons. In the 'on-the-fly' alternative, the architecture is re-evaluated every time it is encountered. We evaluate an architecture with a ranking function 10 times and average the outputs in both cases. Thus, our evaluation is similar to the most typical case of an evolutionary search. To consistently compare evolutionary algorithms we use an evaluation budget and cap it at 1000 architectures. In addition to REA and FreeREA evolutionary searches, we also report results for a greedy evolutionary search. It is similar to REA, but evaluates all possible mutations of the sampled exemplar architecture and selects the best according to the ranking function value. We designed it specially to exclude the randomness introduced by the mutation process. Greedy evolutionary search is only used for benchmarking purposes and is impractical for a real-world setting. See Appendix B for details.

We observe that caching, which is typically applied primarily with the motivation of reducing computational overhead, always improves the search performance. In addition, caching makes the results more stable in the majority of the cases, i.e., reduces the standard deviation. We conjecture that this performance improvement arises because the use of caching means that previous search decisions are not overturned. In the case of 'on-the-fly' evaluation, the search can initially decide that an architecture is inferior, but then

Table 3: Evolutionary search results. Results with averaging (Avg) and statistical (Stat) evaluation are provided. Mann-Whitney U-test is used for the statistical evaluation. Each ranking function is evaluated 10 times for the architecture.

| Search Space | Search Alg. | Eigenvalue score | | ReLU dist | | Ensemble | |
|---|---|---|---|---|---|---|---|
| | | Avg | Stat | Avg | Stat | Avg | Stat |
| NAS-Bench-101 (CIFAR-10) | Greedy | 92.76±0.90 | 92.86±0.66 | 92.36±0.31 | 92.54±0.59 | 92.54±1.01 | 92.82±0.77 |
| | REA | 93.02±0.32 | 93.11±0.30 | 93.15±0.15 | 93.31±0.09 | 92.99±0.42 | 93.28±0.25 |
| | FreeREA | 93.23±0.47 | 93.51±0.35 | 93.29±0.58 | 93.46±0.33 | 93.32±0.53 | 93.52±0.38 |
| NAS-Bench-201 (CIFAR-100) | Greedy | 69.54±0.13 | 70.01±0.42 | 70.00±0.03 | 70.05±0.02 | 71.03±0.46 | 71.12±0.58 |
| | REA | 71.64±0.80 | 72.14±0.69 | 71.83±0.61 | 72.25±0.44 | 72.06±0.64 | 72.40±0.75 |
| | FreeREA | 71.98±0.26 | 73.13±0.07 | 71.63±1.35 | 72.43±1.04 | 71.89±0.97 | 72.15±0.76 |
| NAS-Bench-201 (ImageNet16-120) | Greedy | 42.68±1.06 | 44.08±0.57 | 44.78±1.34 | 45.90±0.71 | 44.11±1.79 | 45.21±0.80 |
| | REA | 44.90±0.76 | 45.18±0.56 | 43.82±1.53 | 44.45±1.06 | 44.23±0.84 | 44.76±0.94 |
| | FreeREA | 44.84±1.17 | 45.52±0.70 | 43.82±1.23 | 44.38±0.92 | 44.26±1.48 | 45.22±0.74 |
| TransNAS-Bench-101 (Jigsaw) | Greedy | 92.49±1.11 | 93.68±0.80 | 92.23±0.94 | 92.98±0.69 | 93.06±0.85 | 94.29±0.41 |
| | REA | 91.88±0.31 | 93.47±0.23 | 92.12±0.39 | 93.29±0.29 | 93.80±0.41 | 95.06±0.60 |
| | FreeREA | 91.79±0.18 | 93.26±0.11 | 92.03±0.17 | 92.97±0.18 | 93.42±0.58 | 94.99±0.58 |
| TransNAS-Bench-101 (Class object) | Greedy | 43.89±1.73 | 45.16±1.44 | 46.04±0.07 | 46.01±0.04 | 46.29±3.34 | 47.19±0.12 |
| | REA | 46.10±0.09 | 46.44±0.07 | 45.99±0.09 | 46.86±0.14 | 46.26±0.08 | 46.52±0.20 |
| | FreeREA | 46.73±0.09 | 46.81±0.08 | 46.97±0.06 | 46.97±0.07 | 47.09±0.11 | 47.14±0.09 |

reverse that decision when the architecture is encountered a second time. That can be clearly seen for a greedy evolutionary search, which is designed to maximise the number of re-evaluations. The efficiency of the search is improved by committing to a decision. The observed performance boost is lower for low-variance ranking functions as their output is more consistent, leading to fewer changed decisions. Table 2 also shows how selection of low variance ranking functions can significantly improve the quality of the search results.

We now compare the performance of the standard averaging approach with that of the proposed statistical evaluation, when each is used in conjunction with evolutionary search. As before, the 5% significance level is used for rejecting the null hypothesis of the Mann-Whitney U-test. We apply caching for both approaches. In the case of averaging, we cache the average of the ranking function output over 10 evaluations. For the statistical evaluation we cache the individual outcomes of the 10 evaluations. Whenever two architectures are statistically compared, we store the outcome, so that the procedure does not need to be repeated.

Table 3 presents the evolutionary search results on the NAS-Bench and TransNAS-Bench search spaces. We demonstrate results on Eigenvalue score and ReLU Hamming distance. We also include an ensemble of those that is formulated as a sum with components individually MinMax normalized over the observed architecture ranks. We observe that statistical evaluation consistently improves the search quality. While the particular improvement depends on the combination of search space and ranking function, we did not observe a case where applying a statistical evaluation leads to poorer search outcome. We observed that FreeREA demonstrates the best results in most of the cases, thus emphasising the importance of crossover in evolutionary search. However, in the averaging case the difference between FreeREA and REA is typically less than one standard deviation. This may make REA more suitable for some practical applications as it requires only to define a mutation operation. FreeREA experiences the largest performance increase with statistical evaluation, and clearly outperforms REA. In some cases, we observed greedy algorithm to outperform others. It is not an expected outcome, but we assume it is due to randomness of experimental setup. Similar to other zero-shot works (Cavagnero et al., 2023a; Wang et al., 2024), our experiments demonstrate that evolutionary search is more stable and efficient than random search. The only search space where random and evolutionary search exhibit similar performance is TransNAS-Bench-101, and we attribute this to the extremely small size of this search space (see Appendix A).

## 6 Conclusion and Future work

In this work, we proposed a statistical approach for comparing architectures during a NAS zero-shot search procedure. Through experiments using both random and evolutionary search, we provided evidence that the proposed approach leads to better search results than simply averaging the outcomes of a batch of evaluations.

We also investigated the variation of a wide selection of zero-shot ranking functions for several search spaces and observed that those which exhibit less variability tend to produce better and more stable search outcomes. Our results also illustrated that caching is beneficial to search, likely because it leads to committing to decisions, allowing for a more efficient search over more architectures. Given these observations, we can recommend that zero-shot search is conducted using caching, low-variability ranking functions, and statistical architecture comparison.

An additional important source of performance variability that we have not discussed – the search space. Our experiments show that the suitability of a ranking function is highly dependent on the search space. For one search space, a particular ranking function might be the best choice; on a different search space, it might perform poorly. Due to this variability, we cannot predict how our conclusions will generalise to a new search space.

## References

Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. Zero-Cost Proxies for Lightweight NAS. In *Proc. Int. Conf. Learn. Representations (ICLR)*, May 2021.

Yash Akhauri, J Pablo Munoz, Nilesh Jain, and Ravi Iyer. Evolving zero cost proxies for neural architecture scoring. *arXiv preprint arXiv:2209.07413*, Dec. 2022.

Manfredo Atzori, Arjan Gijsberts, Simone Heynen, Anne-Gabrielle Mittaz Hager, Olivier Deriaz, Patrick Van Der Smagt, Claudio Castellini, Barbara Caputo, and Henning Müller. Building the NinaPro database: A resource for the biorobotics community. In *Proc. IEEE Int. Conf. Biomed. Robot. Biomechatronics (BioRob)*. IEEE, Jun. 2012.

Niccolò Cavagnero, Luca Robbiano, Barbara Caputo, and Giuseppe Averta. FreeREA: Training-free evolution-based architecture search. In *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2023a.

Niccolò Cavagnero, Luca Robbiano, Francesca Pistilli, Barbara Caputo, and Giuseppe Averta. Entropic Score metric: Decoupling topology and size in Training-free NAS. In *Proc. Int. Conf. Comput. Vis. (ICCV)*, Oct. 2023b.

Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *Proc. Int. Conf. Learn. Representations (ICLR)*, May 2021.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, Jul. 2017.

Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Proc. Int. Conf. Learn. Representations (ICLR)*, May 2019.

Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. TransNAS-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proc. IEEE Conf. Comput. Vis. & Pattern Recognit. (CVPR)*, Jun. 2021.

Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with NumPy. *Nature*, Sep. 2020.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2018.

Mojan Javaheripi, Gustavo de Rosa, Subhabrata Mukherjee, Shital Shah, Tomasz Religa, Caio Cesar Teodoro Mendes, Sebastien Bubeck, Farinaz Koushanfar, and Debadeepta Dey. LiteTransformerSearch: Training-free Neural Architecture Search for efficient language models. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Nov. 2022.

Tangyu Jiang, Haodi Wang, and Rongfang Bie. MeCo: Zero-Shot NAS with one data and single forward pass via minimum eigenvalue of correlation. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2024.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, U. Toronto, 2009.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Adv. Neural Inf. Proc. Syst. (NeurIPS)*, Dec. 2019.

Guihong Li, Yuedong Yang, Kartikeya Bhardwaj, and Radu Marculescu. ZiCo: Zero-shot nas via inverse coefficient of variation on gradients. In *Proc. Int. Conf. Learn. Representations (ICLR)*, Apr. 2022.

Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-NAS: A Zero-Shot NAS for high-performance deep image recognition. In *Proc. Int. Conf. Comput. Vision (ICCV)*, Oct. 2021.

Vasco Lopes, Saeid Alirezazadeh, and Luís A Alexandre. EPE-NAS: Efficient performance estimation without training for neural architecture search. In *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, Sep. 2021.

Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18:50–60, Mar. 1947.

Yash Mehta, Colin White, Arber Zela, Arjun Krishnakumar, Guri Zabergja, Shakiba Moradian, Mahmoud Safari, Kaicheng Yu, and Frank Hutter. Nas-bench-suite: Nas evaluation is (now) surprisingly easy. In *Proc. Int. Conf. Learn. Representations (ICLR)*, Apr. 2022.

Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural Architecture Search without training. In *Proc. Int. Conf. Mach. Learn. (ICML)*, Jul. 2021.

Jisoo Mok, Byunggook Na, Ji-Hoon Kim, Dongyoon Han, and Sungroh Yoon. Demystifying the neural tangent kernel from a practical perspective: Can it be trusted for neural architecture search without training? In *Proc. IEEE Conf. Comput. Vis. & Pattern Recognit. (CVPR)*, Jun. 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2019.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proc. Conf. Artif. Intell. (AAAI)*, Jan. 2019.

Pavel Rumiantsev and Mark Coates. Performing neural architecture search without gradients. In *Proc. IEEE Int. Conf. Acoust. Speech, Signal Proc. (ICASSP)*, Jun. 2023.

Aaron Serianni and Jugal Kalita. Training-free neural architecture search for RNNs and transformers. In *Proc. Annu. Meeting Assoc. Comp. Linguistics (ACL)*, Jul. 2023.

Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Adv. Neur. Inf. Proc. Syst (NeurIPS)*, Dec. 2020.

Renbo Tu, Nicholas Roberts, Misha Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. NAS-Bench-360: Benchmarking neural architecture search on diverse tasks. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Nov. 2022.

Jack Turner, Elliot J. Crowley, Michael O'Boyle, Amos Storkey, and Gavin Gray. Blockswap: Fisher-guided block substitution for network compression on a budget. In *Proc. Int. Conf. Learn. Representations (ICLR)*, Jul. 2020.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, Feb. 2020.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *Proc. Int. Conf. Learn. Representations (ICLR)*, Jul. 2020.

Yan Wang, Liangli Zhen, Jianwei Zhang, Miqing Li, Lei Zhang, Zizhou Wang, Yangqin Feng, Yu Xue, Xiao Wang, Zheng Chen, et al. MedNAS: Multi-Scale Training-Free Neural Architecture Search for medical image analysis. *IEEE Trans. Evol. Comput.*, Jan. 2024.

Huan Xiong, Lei Huang, Mengyang Yu, Li Liu, Fan Zhu, and Ling Shao. On the number of linear regions of Convolutional Neural Networks. In *Proc. Int. Conf. Mach. Learn. (ICML)*, Jul. 2020.

Jingjing Xu, Liang Zhao, Junyang Lin, Rundong Gao, Xu Sun, and Hongxia Yang. Knas: green neural architecture search. In *Proc. Int. Conf. Mach. Learn. (ICML)*, Jul. 2021.

Longxing Yang, Yanxin Fu, Shun Lu, Zihao Sun, Jilin Mei, Wenxiao Zhao, and Yu Hu. Sweet Gradient matters: Designing consistent and efficient estimator for Zero-shot Architecture Search. *Neural Netw.*, 168:237–255, Nov. 2023.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proc. Int. Conf. Mach. Learn. (ICML)*, Jun. 2019.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proc. IEEE Conf. Comput. Vis. & Pattern Recognit. (CVPR)*, Jun. 2018.

## A NAS search space benchmarks

**NAS-Bench-101** Ying et al. (2019) includes the performance and training statistic of the 423k architectures on CIFAR-10. It has three cells. Each cell containing 7 nodes each with a pool of operations consists of: "3x3 conv", "1x1 conv" and "3x3 max-pooling".

**NAS-Bench-201** (Dong & Yang, 2019) has 15625 architectures in the search space and provides performance for CIFAR-10, CIFAR-100, and ImageNet-16-120. It has 5 cells. Every cell has 4 nodes. The list of operations includes "zeroize", which is equivalent to cutting the connection, "skip connection", "3x3 conv", "1x1 conv" and "3x3 avg-pooling". This is the most popular benchmark as it is relatively small and contains measurements for several datasets. **NAS-Bench-360** (Tu et al., 2022) extends this benchmark by additional datasets. In this work, we are using the NinaPro DB5 dataset from NAS-Bench-360. For consistency, we refer to it as NAS-Bench-201 (NinaPro).

**TransNAS-Bench-101** (Duan et al., 2021) is a tabular benchmark that consists of two search spaces evaluated on seven tasks. The architecture of the neural network in the benchmark is a composition of a searchable encoder and a decoder tailored to specific tasks. The macro search space suggests searching for the number of blocks, the size for downsampling, and the count of output channels per block. The micro search space is composed of four nodes, similarly to NAS-Bench-201. Four operations are used: "zeroize", "skip connection", "3x3 conv block" and "1x1 conv block", where every convolution block is a sequence of ReLU, convolution, and batch normalisation. In our project, we are only using the micro search space that consists of 4096 graphs.

## B Evolutionary search algorithms

Here we present pseudocode for all evolutionary algorithms used in this project. Since the greedy evolutionary algorithm and FreeREA are primarily based on REA, we provide full pseudocode for REA (see Algorithm 2) and parts that are different for FreeREA (see Algorithm 4) and greedy evolutionary search (see Algorithm 3). The core notation for all the algorithms is chosen to be close to REA notation, as presented by Real et al. (2019). For the standard implementation, the highest ranked candidate and the top-2 are computed by

**Algorithm 2** Regularised evolutionary algorithm (REA)

$population \leftarrow \emptyset$
$history \leftarrow \emptyset$
**while** $|population| < P$ **do** ▷ Initialize population of size $P$
  $arch \leftarrow$ ArchSamle() ▷ Randomly sample architecture
  $rank \leftarrow$ EvalArch($arch$) ▷ Evaluate ranking function for architecture
  $history \leftarrow history \cup \{(arch, rank)\}$
  $population \leftarrow population \cup \{(arch, rank)\}$
**end while**
**while** $|history| < C$ **do** ▷ Evolve for $C$ cycles
  $random\_candidates \leftarrow \emptyset$
  **for** $i \leftarrow 1, S$ **do**
    $candidate \leftarrow$ Random(population) ▷ Sample random element from population
    $random\_candidates \leftarrow random\_candidates \cup \{candidate\}$
  **end for**
  $parent \leftarrow$ highest rank candidate out of $random\_candidates$
  $child \leftarrow$ Mutate($parent$) ▷ Sample one of all possible mutations
  $rank \leftarrow$ EvalArch($child$) ▷ Evaluate ranking function for the child
  $history \leftarrow history \cup \{(child, rank)\}$
  $population \leftarrow population \cup \{(child, rank)\}$
  remove the oldest member of the population
**end while**
$best \leftarrow$ highest rank candidate out of $history$
**return** $best$

**Algorithm 3** Greedy evolutionary search algorithm difference with Algorithm 2

...
$parent \leftarrow$ highest rank candidate out of $random\_candidates$
$children \leftarrow$ MutateAll($parent$) ▷ Get list of all possible mutations
$child \leftarrow$ highest rank candidate out of $children$
$rank \leftarrow$ EvalArch($child$) ▷ Evaluate all children and get the highest rank one
add all mutations and their respective ranks to the $history$
/* Proceed to line 19 of Algorithm 2 */
...

**Algorithm 4** Free regularised evolutionary algorithm (FreeREA) difference with Algorithm 2

...
$parent_1, parent_2 \leftarrow$ top-2 highest rank candidates out of $random\_candidates$
$child_1 \leftarrow$ Mutate($parent_1$)
$rank_1 \leftarrow$ EvalArch($child_1$)
$child_2 \leftarrow$ Mutate($parent_2$)
$rank_2 \leftarrow$ EvalArch($child_2$)
$child_3 \leftarrow$ Crossover($parent_1, parent_2$) ▷ Perform crossover between two parents' genes
$rank_3 \leftarrow$ EvalArch($child_3$)
$history \leftarrow history \cup \{(child_1, rank_1), (child_2, rank_2), (child_3, rank_3)\}$
$population \leftarrow population \cup \{(child_1, rank_1), (child_2, rank_2), (child_3, rank_3)\}$
remove top-3 the oldest member of the population
/* Proceed to line 21 of Algorithm 2 */
...

Table 4: Random search on $N$ sampled architectures; coefficient of variation (CV) is used as a ranking function. Statistical evaluation of the ranking function (Stat) is provided for the reference. Each ranking function is evaluated 10 times for the architecture on a batch of 64. We provide mean values and standard deviations over 100 runs.

| Search Space | Eigenvalue score | | ReLU dist | | NTK(cond) | |
|---|---|---|---|---|---|---|
| | CV | Stat | CV | Stat | CV | Stat |
| **N**= 10 | | | | | | |
| NAS-Bench-101 (CIFAR-10) | 91.30±3.04 | 91.71±2.22 | 91.74±1.74 | 91.63±1.51 | 91.15±1.68 | 91.24±1.63 |
| NAS-Bench-201 (ImageNet16-120) | 39.58±6.11 | 41.22±4.56 | 37.02±5.80 | 41.24±3.33 | 42.59±4.73 | 41.64±4.76 |
| TransNAS-Bench-101 (Class object) | 47.24±5.00 | 48.26±2.84 | 48.72±2.32 | 48.71±2.69 | 46.72±5.44 | 48.48±2.46 |
| **N**= 100 | | | | | | |
| NAS-Bench-101 (CIFAR-10) | 91.36±1.56 | 92.06±1.15 | 91.61±2.68 | 91.98±2.01 | 90.18±1.83 | 90.06±1.98 |
| NAS-Bench-201 (ImageNet16-120) | 35.98±5.59 | 40.96±4.98 | 35.97±5.92 | 43.34±2.47 | 38.21±5.29 | 41.50±4.42 |
| TransNAS-Bench-101 (Class object) | 45.27±6.20 | 49.46±4.56 | 45.30±6.37 | 48.87±2.96 | 48.24±2.77 | 49.81±1.77 |
| **N**= 1000 | | | | | | |
| NAS-Bench-101 (CIFAR-10) | 91.21±1.51 | 91.57±1.31 | 92.64±1.51 | 92.60±1.73 | 89.04±1.89 | 90.92±2.40 |
| NAS-Bench-201 (ImageNet16-120) | 35.26±5.50 | 42.17±4.11 | 35.51±5.35 | 43.79±2.92 | 35.20±5.69 | 42.24±2.03 |
| TransNAS-Bench-101 (Class object) | 44.79±6.72 | 46.46±4.21 | 47.39±3.83 | 47.82±5.97 | 45.20±6.40 | 49.36±0.89 |

maximizing over the averaged ranking function output. For our implementation, we use statistical comparison with stat-max and stat-top-2 from Algorithm 1.

We designed a greedy evolutionary search algorithm in order to further demonstrate how ranking function variance influences evolutionary search algorithm performance. The primary difference between greedy evolutionary search and REA lies in the mutation operation. Instead of randomly mutating a parent candidate as in REA, it greedily explores a list of all possible mutations. Therefore, it is expected to experience a greater impact from the variance introduced by a ranking function.

In all our experiments, we are utilising an evaluation budget ($C$ at line 9 of Algorithm 2) that defines how many architectures can be evaluated in total. As one can see, the three evolutionary algorithms are spending the budget at different rates. REA evaluates a single candidate per iteration. FreeREA evaluates three and greedy search evaluates all the mutations (i.e., the neighbourhood on the search hypergraph).

## C Coefficient of variations as a ranking function

A naive way to incorporate a coefficient of variation into the search process is to utilise it as a ranking function. Given a ranking function $r(arch, d)$ we construct a coefficient of variation estimator as

$$r_{CV}(arch, \mathcal{D}) = Mean(\mathcal{M}) + CV(\mathcal{M}) \quad \text{for} \quad \mathcal{M} = \{r(arch, d_k)\}_{k=1}^{K}\,, \tag{2}$$

where $d_k \in \mathcal{D}$ is a batch of data and $CV(\mathcal{X}) = \frac{Var(\mathcal{X})}{Mean(\mathcal{X})}$.

As shown in Fig. 3, the ranking function coefficient of variation exhibits a considerable Kendall-$\tau$ correlation with accuracy. Moreover, in almost all cases this correlation is negative, which makes coefficient of variation a suitable ranking function for many metrics.

To explore this further, we conduct an experiment for several search spaces where we use random search and sample $N$ architectures. To stabilize search performance, we use Min-Max normalization individually for the mean term and the CV term when computing $r_{CV}$ in equation 2.

We demonstrate some results in Table 4. Although in some cases on NAS-Bench-101 $r_{CV}$ exhibits performance comparable to statistical evaluation, its performance is consistently worse on other search spaces, in some cases by a considerable margin. In many cases, the ranking function modification is harmful for the performance compared to just using the mean. We attribute this to the correlation between the coefficient of variation and a ranking function averaged value (see Fig. 4). We have noticed that the Kendall-$\tau$ correlation between the coefficient of variation of a ranking function and its mean value is the highest for the low-variance ranking functions that are the most suitable for zero-shot search (see Fig. 4). For those ranking

functions, the coefficient of variation does not provide much additional useful information for the search, but can introduce additional variability, thus making the results inconsistent.

## D Assets

In this work, we used the following software:

- *automl/NASLib* (Mehta et al., 2022) available on GitHub under Apache 2.0 Licence
- *PyTorch* (Paszke et al., 2019) available via PyPI under a custom BSD licence
- *NumPy* (Harris et al., 2020) available via PyPI under a custom BSD licence
- *SciPy* (Virtanen et al., 2020) available via PyPI under a custom BSD licence

The following dataset were used:

- *CIFAR-10/100* (Krizhevsky, 2009) under CC BY 4.0 Licence
- *ImageNet-16-120* (Chrabaszcz et al., 2017) under CC BY 4.0 Licence
- *NinaPro* (Atzori et al., 2012) under CC BY-ND Licence
- Five datasets from *Taskonomy collection* (Zamir et al., 2018) under CC BY 4.0 Licence