NAS-Bench-Suite-Zero: Accelerating Research on Zero Cost Proxies

Arjun Krishnakumar[*1], **Colin White**[*2], **Arber Zela**[*1], **Renbo Tu**[*3],
Mahmoud Safari[1], **Frank Hutter**[1,4]

[1]University of Freiburg, [2]Abacus.AI, [3]University of Toronto,
[4]Bosch Center for Artificial Intelligence

Abstract

Zero-cost proxies (ZC proxies) are a recent architecture performance prediction technique aiming to significantly speed up algorithms for neural architecture search (NAS). Recent work has shown that these techniques show great promise, but certain aspects, such as evaluating and exploiting their complementary strengths, are under-studied. In this work, we create `NAS-Bench-Suite-Zero`: we evaluate 13 ZC proxies across 28 tasks, creating by far the largest dataset (and unified codebase) for ZC proxies, enabling orders-of-magnitude faster experiments on ZC proxies, while avoiding confounding factors stemming from different implementations. To demonstrate the usefulness of `NAS-Bench-Suite-Zero`, we run a large-scale analysis of ZC proxies, including a bias analysis, and the first information-theoretic analysis which concludes that ZC proxies capture substantial complementary information. Motivated by these findings, we present a procedure to improve the performance of ZC proxies by reducing biases such as cell size, and we also show that incorporating all 13 ZC proxies into the surrogate models used by NAS algorithms can improve their predictive performance by up to 42%. Our code and datasets are available at `https://github.com/automl/naslib/tree/zerocost`.

1 Introduction

Algorithms for neural architecture search (NAS) seek to automate the design of high-performing neural architectures for a given dataset. NAS has successfully been used to discover architectures with better accuracy/latency tradeoffs than the best human-designed architectures [5, 9, 28, 38]. Since early NAS algorithms were prohibitively expensive to run [58], a long line of recent work has focused on improving the runtime and efficiency of NAS methods (see [9, 49] for recent surveys).

A recent thread of research within NAS focuses on *zero-cost proxies* (ZC proxies) [1, 23]. These novel techniques aim to give an estimate of the (relative) performance of neural architectures from just a *single minibatch of data*. Often taking just five seconds to run, these techniques are essentially "zero cost" compared to training an architecture or to any other method of predicting the performance of neural architectures [48]. Since the initial ZC proxy was introduced [23], there have been many follow-up methods [1, 16]. However, several recent works have shown that simple baselines such as "number of parameters" and "FLOPS" are competitive with all existing ZC proxies across most settings, and that most ZC proxies do not generalize well across different benchmarks, thus requiring broader large-scale evaluations in order to assess their strengths [2, 25]. A recent landscape overview concluded that ZC proxies show great promise, but certain aspects are under-studied and their true

[*]Equal contribution. Work done while RT was part-time at Abacus.AI. Email to:
{krishnan, zelaa, fh}@cs.uni-freiburg.de, colin@abacus.ai, renbo.tu@mail.utoronto.ca, safarim@informatik.uni-freiburg.de.

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks.

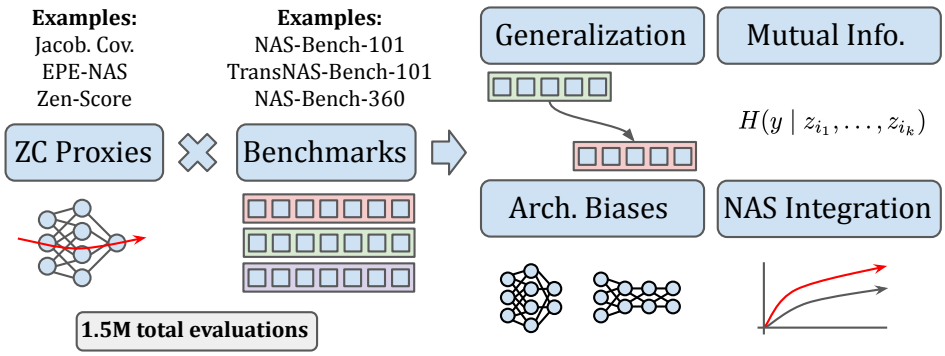

Figure 1: Overview of `NAS-Bench-Suite-Zero`. We implement and pre-compute 13 ZC proxies on 28 tasks in a unified framework, and then use this dataset to analyze the generalizability, complementary information, biases, and NAS integration of ZC proxies.

potential has not been realized thus far [45]. In particular, it is still largely unknown whether ZC proxies can be effectively combined, and how best to integrate ZC proxies into NAS algorithms.

In this work, we introduce `NAS-Bench-Suite-Zero`: a unified and extensible collection of 13 ZC proxies, accessible through a unified interface, which can be evaluated on a suite of 28 tasks through `NASLib` [30] (see Figure 1). In addition to the codebase itself, we release precomputed ZC proxy scores across all 13 ZC proxies and 28 tasks, which can be used to speed up ZC proxy experiments. Specifically, we show that the runtime of ZC proxy experiments such as NAS analyses and bias analyses are shortened by a factor of at least 10^3 when using the precomputed ZC proxies in `NAS-Bench-Suite-Zero`. By providing a unified framework with ready-to-use scripts to run large-scale experiments, `NAS-Bench-Suite-Zero` eliminates the overhead for researchers to compare against many other methods and across all popular NAS benchmark search spaces, helping the community to rapidly increase the speed of research in this promising direction. Our benchmark suite was very recently used successfully in the Zero Cost NAS Competition at AutoML-Conf 2022. See Appendix F for more details. In Appendix B, we give detailed documentation, including a datasheet [10], license, author responsibility, code of conduct, and maintenance plan. We welcome contributions from the community and hope to grow the repository and benchmark suite as more ZC proxies and NAS benchmarks are released.

To demonstrate the usefulness of `NAS-Bench-Suite-Zero`, we run a large-scale analysis of ZC proxies: we give a thorough study of generalizability and biases, and we give the first information-theoretic analysis. Interestingly, based on the bias study, **we present a concrete method for improving the performance of a ZC proxy by reducing biases** (such as the tendency to favor larger architectures or architectures with more `conv` operations). This may have important consequences for the future design of ZC proxies. Furthermore, based on the information-theoretic analysis, we find that there is high information gain of the validation accuracy when conditioned on multiple ZC proxies, suggesting that ZC proxies do indeed compute substantial complementary information. Motivated by these findings, we incorporate all 13 proxies into the surrogate models used by NAS algorithms [44, 47], showing that the Spearman rank correlation of the surrogate predictions can *increase by up to 42%*. We show that this results in improved performance for two predictor-based NAS algorithms: BANANAS [47] and NPENAS [44].

Our contributions. We summarize our main contributions below.

- We release `NAS-Bench-Suite-Zero`, a collection of benchmarks and ZC proxies that unifies and accelerates research on ZC proxies – a promising new sub-field of NAS – by enabling orders-of-magnitude faster evaluations on a large suite of diverse benchmarks.
- We run a large-scale analysis of 13 ZC proxies across 28 different combinations of search spaces and tasks by studying the generalizability, bias, and mutual information among ZC proxies.
- Motivated by our analysis, we present a procedure to improve the performance of ZC proxies by reducing biases, and we show that the complementary information of ZC proxies can significantly improve the predictive power of surrogate models commonly used for NAS.

Table 1: List of ZC proxies in `NAS-Bench-Suite-Zero`. Note that "neuron-wise" denotes whether the total score is a sum of individual weights.

Name	Data-dependent	Neuron-wise	Type	In `NAS-Bench-Suite-Zero`
`epe-nas` [21]	✓	✗	Jacobian	✓
`fisher` [42]	✓	✓	Pruning-at-init	✓
`flops` [25]	✓	✓	Baseline	✓
`grad-norm` [1]	✓	✓	Pruning-at-init	✓
`grasp` [43]	✓	✓	Pruning-at-init	✓
`l2-norm` [1]	✗	✗	Baseline	✓
`jacov` [23]	✓	✗	Jacobian	✓
`nwot` [23]	✓	✗	Jacobian	✓
`params` [25]	✗	✓	Baseline	✓
`plain` [1]	✓	✓	Baseline	✓
`snip` [14]	✓	✓	Pruning-at-init	✓
`synflow` [39]	✗	✓	Pruning-at-init	✓
`zen-score` [16]	✗	✗	Piece. Lin.	✓

2 Background and Related Work

Given a dataset and a *search space* – a large set of neural architectures – NAS seeks to find the architecture with the highest validation accuracy (or the best application-specific trade-off among accuracy, latency, size, and so on) on the dataset. NAS has been studied since the late 1980s [24, 40] and has seen a resurgence in the last few years [18, 58], with over 1000 papers on NAS in the last two years alone. For a survey of the different techniques used for NAS, see [9, 49].

Many NAS methods make use of performance prediction. A *performance prediction* method is any function which predicts the (relative) performance of architectures, without fully training the architectures [48]. BRP-NAS [8], BONAS [34], and BANANAS [47] are all examples of NAS methods that make use of performance prediction. While performance prediction speeds up NAS algorithms by avoiding fully training neural networks, many still require non-trivial computation time. On the other hand, a recently-proposed line of techniques, *zero-cost proxies* (ZC proxies) require just a single forward pass through the network, often taking just five seconds [23].

Zero-cost proxies. The original ZC proxy estimated the separability of the minibatch of data into different linear regions of the output space [23]. Many other ZC proxies have been proposed since then, including data-independent ZC proxies [1, 15, 16, 39], ZC proxies inspired by pruning-at-initialization techniques [1, 14, 39, 43], and ZC proxies inspired by neural tangent kernels [4, 35]. See Table 1 for a full list of the ZC proxies we use in this paper. We describe theoretical ZC proxy results in Appendix C.1.

Search spaces and tasks. In our experiments, we make use of several different NAS benchmark search spaces and tasks. NAS-Bench-101 [54] is a popular cell-based search space for NAS research. It consists of 423 624 architectures trained on CIFAR-10. The cell-based search space is designed to model ResNet-like and Inception-like cells [12, 37]. NAS-Bench-201 [6] is a cell-based search space consisting of 15 625 architectures (6 466 non-isomorphic) trained on CIFAR-10, CIFAR-100, and ImageNet16-120. NAS-Bench-301 [56] is a surrogate NAS benchmark for the DARTS search space [19]. The search space consists of normal cell and reduction cells, with 10^{18} total architectures. TransNAS-Bench-101 [7] is a NAS benchmark consisting of two different search spaces: a "micro" (cell-based) search space of size 4 096, and a macro search space of size 3 256. The architectures are trained on seven different tasks from the Taskonomy dataset [55]. NAS-Bench-Suite [22] collects these search spaces and tasks within the unified framework of `NASLib` [30]. In this work, we extend this collection by adding two datasets from NAS-Bench-360 [41], SVHN, and four datasets from Taskonomy. NAS-Bench-360 is a collection of diverse tasks that are ready-to-use for NAS research.

Large-scale studies of ZC proxies. A few recent works [2, 25, 45, 48] investigated the performance of ZC proxies in ranking architectures over different NAS benchmarks, showing that the relative performance highly depends on the search space, but none study more than 12 total tasks, and none make the ZC proxy values publicly available. Two predictor-based NAS methods have recently been introduced: OMNI [48] and ProxyBO [33]. However, OMNI only uses a single ZC proxy, and

Table 2: Overview of ZC proxy evaluations in `NAS-Bench-Suite-Zero`. * Note that EPE-NAS is only defined for classification tasks [21].

Search space	Tasks	Num. ZC proxies	Num. architectures	Total ZC proxy evaluations
NAS-Bench-101	1	13	10 000	130 000
NAS-Bench-201	3	13	15 625	609 375
NAS-Bench-301	1	13	11 221	145 873
TransNAS-Bench-101-Micro	7	12*	3 256	273 504
TransNAS-Bench-101-Macro	7	12*	4 096	344 064
Add'l. 201, 301, TNB-Micro	9	13	600	23400
Total	28	13	44 798	1 526 216

while ProxyBO uses three, the algorithm dynamically chooses one in each iteration (so individual predictions are made using a single ZC proxy at a time). Recently, NAS-Bench-Zero was introduced [2], a new benchmark based on popular computer vision models ResNet [12] and MobileNetV2 [31], which includes 10 ZC proxies. However, the NAS-Bench-Zero dataset is currently not publicly available. For more related work details, see Appendix C.

Only two prior works combine the information of multiple ZC proxies together in architecture predictions [1, 2] and both only use the *voting* strategy to combine at most four ZC proxies. Our work is the first to publicly release ZC proxy values, combine ZC proxies in a nontrivial way, and exploit the complementary information of 13 ZC proxies simultaneously.

3 Overview of NAS-Bench-Suite-Zero

In this section, we give an overview of the `NAS-Bench-Suite-Zero` codebase and dataset, which allows researchers to quickly develop ZC proxies, compare against existing ZC proxies across diverse datasets, and integrate them into NAS algorithms, as shown in Sections 4 and 5.

We implement all ZC proxies from Table 1 in the same codebase (`NASLib` [30]). For all ZC proxies, we use the default implementation from the original work. While this list covers 13 ZC proxies, the majority of ZC proxies released to date, we did not yet include a few other ZC proxies, for example, due to requiring a trained supernetwork to make evaluations [4, 35] (therefore needing to implement a supernetwork on 28 benchmarks), implementation in TensorFlow rather than PyTorch [26], or unreleased code. Our modular framework easily allows additional ZC proxies to be added to `NAS-Bench-Suite-Zero` in the future.

To build `NAS-Bench-Suite-Zero`, we extend the collection of `NASLib`'s publicly available benchmarks, known as NAS-Bench-Suite [22]. This allows us to evaluate and fairly compare all ZC proxies in the same framework without confounding factors stemming from different implementations, software versions or training pipelines. Specifically, for the search spaces and tasks, we use NAS-Bench-101 (CIFAR-10), NAS-Bench-201 (CIFAR-10, CIFAR-100, and ImageNet16-120), NAS-Bench-301 (CIFAR-10), and TransNAS-Bench-101 Micro and Macro (Jigsaw, Object Classification, Scene Classification, Autoencoder) from NAS-Bench-Suite. We add the remaining tasks from TransNAS-Bench-101 (Room Layout, Surface Normal, Semantic Segmentation), and three tasks each for NAS-Bench-201, NAS-Bench-301, and TransNAS-Bench-101-Micro: Spherical-CIFAR-100, NinaPro, and SVHN. This yields a total of 28 benchmarks in our analysis. For all NAS-Bench-201 and TransNAS-Bench-101 tasks, we evaluate all ZC proxy values and the respective runtimes, for all architectures. For NAS-Bench-301, we evaluate on all 11 221 randomly sampled architectures from the NAS-Bench-301 dataset, due to the computational infeasibility of exhaustively evaluating the full set of 10^{18} architectures. Similarly, we evaluate 10 000 architectures from NAS-Bench-101. Finally, for Spherical-CIFAR-100, NinaPro, and SVHN, we evaluate 200 architectures per search space, since only 200 architectures are fully trained for each of these tasks. See Table 2.

We run all ZC proxies from Table 1 on Intel Xeon Gold 6242 CPUs and save their evaluations in order to create a queryable table with these pre-computed values. We use a batch size of 64 for all ZC proxy evaluations, except for the case of TransNAS-Bench-101: due to the extreme memory usage of the Taskonomy tasks ($>$ 30GB memory), we used a batch size of 32. The total computation time for all 1.5M evaluations was 1100 CPU hours.

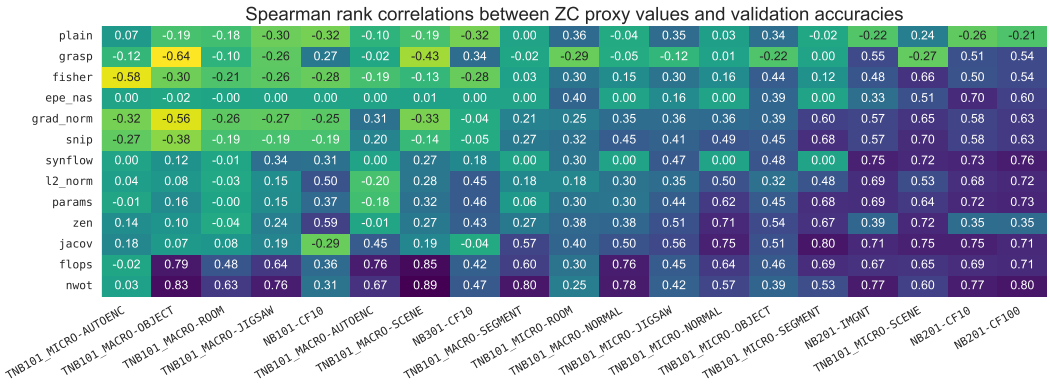

Figure 2: Spearman rank correlation coefficient between ZC proxy values and validation accuracies, for each ZC proxy and benchmark. The rows and columns are ordered based on the mean scores across columns and rows, respectively.

Speedups and recommended usage. The average time to compute a ZC proxy across all tasks is 2.6 seconds, and the maximum time (computing `grasp` on TNB-Macro Autoencoder) is 205 seconds, compared to 10^{-5} seconds when instead querying the `NAS-Bench-Suite-Zero`API.

When researchers evaluate ZC proxy-based NAS algorithms using queryable NAS benchmarks, the bottleneck is often (ironically) the ZC proxy evaluations. For example, for OMNI [48] or ProxyBO [33] running for 100 iterations and 100 candidates per iteration, the total evaluation time is roughly 9 hours, yet they can be run on `NAS-Bench-Suite-Zero` in under one minute. Across all experiments done in this paper (mutual information study, bias study, NAS study, etc.), we calculate that using `NAS-Bench-Suite-Zero` decreases the computation time by at least three orders of magnitude. See Appendix D.4 for more details.

Since `NAS-Bench-Suite-Zero` reduces the runtime of experiments by at least three orders of magnitude (on queryable NAS benchmarks), we recommend researchers take advantage of `NAS-Bench-Suite-Zero` to *(i)* run hundreds of trials of ZC proxy-based NAS algorithms, to reach statistically significant conclusions, *(ii)* run extensive ablation studies, including the type and usage of ZC proxies, and *(iii)* increase the total number of ZC proxies evaluated in the NAS algorithm. Finally, when using `NAS-Bench-Suite-Zero`, researchers should report the real-world time NAS algorithms would take, by adding the time to run each ZC proxy evaluation (which can be queried in `NAS-Bench-Suite-Zero`) to the total runtime of the NAS algorithm.

4 Generalizability, Mutual Information, and Bias of ZC Proxies

In this section, we use `NAS-Bench-Suite-Zero` to study concrete research questions relating to the generalizability, complementary information, and bias of ZC proxies.

4.1 RQ 1: How well do ZC proxies generalize across different benchmarks?

In Figure 2, for each ZC proxy and each benchmark, we compute the Spearman rank correlation between the ZC proxy values and the validation accuracies over a set of 1000 randomly drawn architectures (see Appendix D for the full results on all benchmarks). Out of all the ZC proxies, `nwot` and `flops` have the highest rank correlations across all benchmarks. On some of the benchmarks, such as TransNAS-Bench-101-Micro Autoencoder and Room Layout, all of the ZC proxies exhibit poor performance on average, while on the widely used NAS-Bench-201 benchmarks, almost all of them perform well. Several methods, such as `snip` and `grasp`, perform well on the NAS-Bench-201 tasks, but on average are outperformed by `params` and `flops` on the other benchmarks.

Although no ZC proxy performs consistently across all benchmarks, we may ask a related question: is the performance of all ZC proxies across benchmarks correlated enough to capture similarities among benchmarks? In other words, can we use ZC proxies as a tool to assess the similarities among tasks. This is particularly important in meta-learning or transfer learning, where a meta-algorithm aims to learn and transfer knowledge across a set of similar tasks. To answer this question, we

compute the Pearson correlation of the ZC proxy scores on each pair of benchmarks. See Figure 3. As expected, benchmarks that are based on the same or similar search spaces are highly correlated with respect to the ZC proxy scores. For example, we see clusters of high correlation for the Trans-NAS-Bench-101-Macro benchmarks, and the NAS-Bench-201 benchmarks.

Answer to RQ 1: *Only a few ZC proxies generalize well across most benchmarks and tasks. However, ZC proxies can be used to assess similarities across benchmarks.* This suggests the potential future direction of incorporating them as task features in a meta-learning setting [20].

4.2 RQ 2: Are ZC proxies complementary with respect to explaining validation accuracy?

While Figure 2 shows the performance of each individual ZC proxy, now we consider the combined performance of multiple ZC proxies. If ZC proxies measure different characteristics of architectures, then a NAS algorithm can exploit their complementary information in order to yield improved results. While prior work [25, 45] computes the correlation among pairs of ZC proxies, [2] our true goal is to assess the complementary information of ZC proxies *with respect to explaining the ground-truth validation accuracy*. Furthermore, we wish to measure the complementary information of more than just two ZC proxies at a time. For this, we turn to information theoretic measures: by treating the validation accuracy and ZC proxy values as random variables, we can measure the entropy of the validation accuracy conditioned on one or more ZC proxies, which intuitively tells us the information that one or more ZC proxies reveal about the validation accuracy.

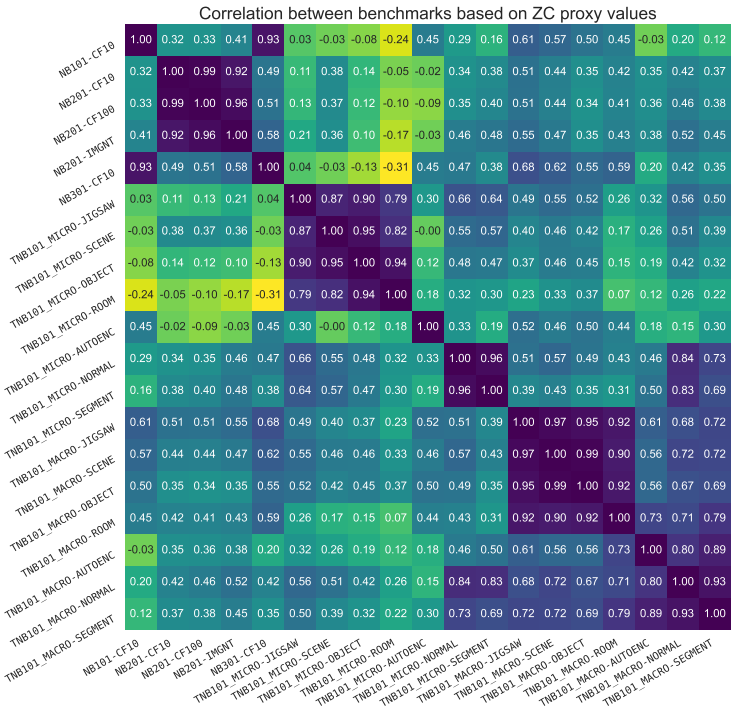

Figure 3: Pearson correlation coefficient between ZC proxy scores on pairs of benchmarks. The entries in the plot are ordered based on the mean score across each row and column.

Formally, given a search space S, let $\mathcal{Y}$ denote the uniform distribution of validation accuracies over the search space, and let y denote a random sample from $\mathcal{Y}$. Similarly, for a ZC proxy i from 1 to 13, let $\mathcal{Z}_i$ denote the uniform distribution of the ZC proxy values, and let z_i denote a random sample from $\mathcal{Z}_i$. Let $H(\cdot)$ denote the entropy function. For all pairs z_i, z_j of ZC proxies, we compute the conditional entropy $H(y \mid z_i, z_j)$, as well as the information gain $H(y \mid z_i) - H(y \mid z_i, z_j)$. See Figure 4. The entropy computations are based on 1000 randomly sampled architectures, using 24-bin histograms for density smoothing (see Appendix D for more details). We see that `synflow` and `plain` together give the most information about the ground truth validation accuracies, due to their substantial complementary information.

Now we can ask the same question for k tuples of ZC proxies. Given an ordered list of k ZC proxies $z_{i_1}, z_{i_2}, \ldots z_{i_k}$, we define the information gain of z_{i_k} conditioned on y as follows:

$$\textbf{IG}(z_{i_k}) \coloneqq H(y \mid z_{i_1}, \ldots, z_{i_{k-1}}) - H(y \mid z_{i_1}, \ldots, z_{i_k}). \tag{1}$$

Intuitively, **IG** computes the marginal information we learn about y when z_{i_k} is revealed, assuming we already knew the values of $z_{i_1}, \ldots, z_{i_{k-1}}$. We compare the conditional entropy vs. number of

[2] For completeness, we re-run that experiment and include the results in Appendix D.

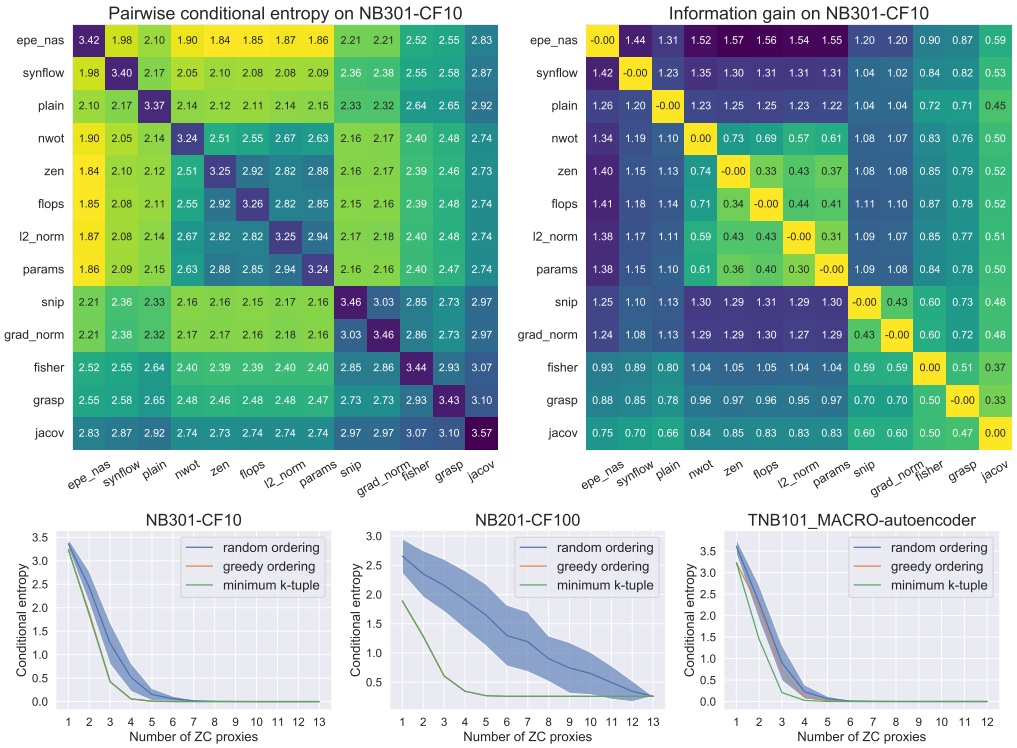

Figure 4: Given a ZC proxy pair (i, j), we compute the conditional entropy $H(y \mid z_i, z_j)$ (top left), and information gain $H(y \mid z_i) - H(y \mid z_i, z_j)$ (top right). Conditional entropy $H(y \mid z_{i_1}, \ldots, z_{i_k})$ vs. k, where the ordering $z_{i_1}, \ldots, z_{i_k}$ is selected using three different strategies. The minimum k-tuple and greedy ordering significantly overlap in the first two figures (bottom).

ZC proxies for three different orderings of the ZC proxies. The first is a random ordering (averaged over 100 random trials), which tells us the average information gain when iteratively adding more ZC proxies. The second is a greedy ordering, computed by iteratively selecting the ZC proxy that maximizes $\mathbf{IG}(z_{i_k})$, for k from 1 to 13. The final plot exhaustively searches through $\binom{13}{k}$ sets to find the k proxies which minimize $H(y \mid z_{i_1}, \ldots z_{i_k})$, for k from 1 to 13 (note that this may not define a valid ordering). See Figure 4, and Appendix D for the complete results. We see that there is very substantial information gain when iteratively adding ZC proxies, even if the ZC proxies are randomly chosen. Optimizing the order of adding ZC proxies yields much higher **IG** in certain benchmarks (e.g., NB201-CF100), and a greedy approach is shown to be not far from the optimum.

Answer to RQ 2: *In some benchmarks, we see substantial complementary information among ZC proxies. However, the degree of complementary information depends heavily on the NAS benchmark at hand.* This suggests that we cannot always expect ZC proxies to yield complementary information, but a machine learning model might be able to identify useful combinations of ZC proxies.

4.3 RQ 3: Do ZC proxies contain biases, such as a bias toward certain operations or sizes, and can we mitigate these biases?

Identifying biases in ZC proxies can help explain weaknesses and facilitate the development of higher-performing ZC proxies. We define bias metrics and study ZC proxy scores for thousands of architectures for their correlation with biases. This systematic approach yields generalizable conclusions and avoids the noise from assessing singular architectures. We consider the following biases: **conv:pool** (the numerical advantage of convolution to pooling operations in the cell), **cell size** (the number of non-zero operations in the cell), **num. skip connections**, and **num. parameters**.

For each search space, ZC proxy, and bias, we compute the Pearson correlation coefficient between the ZC proxy values and the bias values. We consider all 44K architectures referenced in Table 2. See Table 3 and Appendix D for the full results. We find that many ZC proxies exhibit biases to

Table 3: Pearson correlation coefficients between predictors and bias metrics (in bold) on different datasets. For example, for **Cell size** on NB201-CF100, `snip` has a correlation of -0.04 (indicating very little bias), while `synflow` has a correlation of 0.57 (meaning it favors larger architectures).

Name	Conv:pool		Cell size		Num. skip connections		Num. parameters	
	NB201-CF10	NB301-CF10	NB201-CF100	NB201-IM	NB301-CF10	NB201-CF100	NB101-CF10	NB301-CF10
epe-nas	0.05	-0.02	0.35	0.35	0.01	0.09	-0.02	-0.01
fisher	0.05	0.01	-0.03	-0.05	-0.15	-0.03	0.11	0.17
flops	0.59	0.70	0.30	0.30	**-0.35**	-0.30	**1.00**	0.99
grad-norm	0.35	0.27	-0.04	-0.05	-0.26	-0.26	0.30	0.51
grasp	0.01	0.28	-0.01	0.01	0.03	0.00	-0.03	0.24
l2-norm	**0.87**	0.76	0.41	0.41	-0.33	**-0.41**	0.62	0.99
jacov	0.05	-0.11	0.35	0.35	0.08	0.09	-0.18	-0.10
nwot	0.06	**0.78**	0.28	0.28	-0.21	0.06	0.74	0.95
params	0.61	**0.78**	0.29	0.29	-0.32	-0.29	**1.00**	**1.00**
plain	-0.33	-0.45	0.14	0.14	0.02	0.02	0.03	-0.45
snip	0.37	0.27	-0.04	-0.04	-0.28	-0.28	0.44	0.50
synflow	0.53	0.41	**0.57**	**0.58**	-0.20	-0.14	0.57	0.62
zen-score	0.05	0.75	0.35	0.35	-0.33	0.09	0.68	0.99
val-acc	0.36	0.45	0.35	0.43	0.13	-0.06	0.09	0.47

various degrees. Interestingly, some biases are consistent across search spaces, while others are not. For example, `l2-norm` has a conv:pool bias on both NB201-C10 and NB301-C10, while `nwot` has a strong conv:pool bias on NB301-C10 and almost no bias on NB201-C10. While validation accuracy does not correlate with number of skip connections, most ZC proxies in the benchmark exhibit a negative bias towards this metric.

Next, we present a procedure for removing these biases. For this study, we use ZC proxies that had large biases in Table 3, and we attempt to answer the following questions: *(1)* can we remove these biases, and *(2)* if we can remove the biases, does the performance of ZC proxies improve?

Given a search space of architectures A, let $f : A \to \mathbb{R}$ denote a ZC proxy (a function that takes as input an architecture, and outputs a real number). Furthermore, let $b : A \to \mathbb{R}$ denote a bias measure such as "cell size". Recall that Table 3 showed that the correlation between a ZC proxy f and a bias measure b may be high. For example, the correlation between `synflow` and "cell size" is high, which means using `synflow` would favor larger architectures. To reduce bias, we use a simple heuristic:

$$f'(a) = f(a) \cdot \frac{1}{b(a) + C}. \tag{2}$$

In this expression, C is a constant that we can tune. In deciding on a strategy to tune C, we make two observations. First, for most bias measures, the bias of `val_acc` is not zero, which means completely de-biasing ZC proxies could hurt performance. Second, depending on the application, we may want to fully remove the bias of a ZC proxy, or else remove bias only insofar as it improves performance.

Therefore, we test three different strategies to tune C by brute force: *(1)* "minimize", to minimize bias, *(2)* "equalize", to match the bias with the bias of `val_acc`, and *(3)* "performance", to optimize the performance (Pearson correlation). See Table 4 for the results.

We find that using the "performance" strategy, we are able to increase the performance of ZC proxies by reducing their bias. Furthermore, the "equalize" strategy sometimes provide good results on par with the "performance" strategy. This suggests a good bias mitigation strategy when we do not know the ground truth but have information on how the ground truth correlations with bias. This may have important consequences for the future design of ZC proxies.

Answer to RQ 3: *Many ZC proxies do exhibit different types of biases to various degrees, but the biases can be mitigated, thereby improving performance.*

5 Integration into NAS

The findings in Section 4.2 showed that ZC proxies contain substantial complementary information, conditioned on the ground-truth validation accuracies. However, no prior work has combined more than four ZC proxies, or used a combination strategy other than a simple vote. In this section, we combine and integrate all 13 ZC proxies into predictor-based NAS algorithms by adding the ZC proxies directly as features into the surrogate (predictor) models.

Table 4: Bias mitigation strategies tested on the ZC proxies with the most biases. We test three different strategies by tuning C from Equation 2 for different objectives: minimize (tune C to minimize bias), equalize (tune C to match ground truth's correlation with bias metric), and performance (tune C to maximize correlation with ground truth). Bias and performance are Pearson correlation coefficients of the proxy score with the bias metric and with the ground truth accuracy, respectively. C is searched between -10 and 1000.

ZC proxy	dataset	bias metric	original bias	original perf.	new bias	new perf.	strategy
l2-norm	NB201-CF10	conv:pool	0.87	0.42	0.00	0.10	minimize
					0.37	0.11	equalize
					0.70	0.44	performance
nwot	NB301-CF10	conv:pool	0.78	0.49	0.00	0.03	minimize
					0.29	0.14	equalize
					0.78	0.49	performance
synflow	NB201-CF100	cell size	0.57	0.68	0.01	0.64	minimize
					0.35	0.71	equalize
					0.35	0.71	performance
synflow	NB201-IM	cell size	0.58	0.76	0.01	0.62	minimize
					0.43	0.76	equalize
					0.46	0.76	performance
flops	NB301-CF10	num. skip	-0.35	0.43	-0.01	0.06	minimize
					0.12	-0.05	equalize
					-0.35	0.43	performance

We run experiments on two common predictor-based NAS algorithms: BANANAS, based on Bayesian optimization [47], and NPENAS, based on evolution [44]. Both algorithms use a model-based performance predictor: a model that takes in an architecture encoding as features (e.g., the adjacency matrix encoding [46]), and outputs a prediction of that architecture's validation accuracy. The model is retrained throughout the search algorithm, as more and more architectures are fully trained. Recent work has shown that boosted trees such as XGBoost achieve strong performance in NAS [48, 56].

Experimental setup. For both algorithms, we use the `NASLib` implementation [30] and default parameters reported in prior work [48]. First, we assess the standalone performance of XGBoost when ZC proxies are added as features in addition to the architecture encoding, by randomly sampling 100 training architectures and 1000 disjoint test architectures, and computing the Spearman rank correlation coefficient between the set of predicted validation accuracies and the ground-truth accuracies. On NAS-Bench-201 CIFAR-100, averaged over 100 trials, the Spearman rank correlation ($\pm$ std. dev.) improves from 0.640 ± 0.0420 to $\mathbf{0.908 \pm 0.012}$ with the addition of ZC proxies, representing an *improvement of 41.7%*. Even more surprisingly, using the ZC proxies alone as features without the architecture, results in a Spearman rank correlation of $\mathbf{0.907 \pm 0.013}$, implying that the ZC proxies subsume nearly all information contained in the architecture encoding itself. We present the full results in Appendix E. These results show that an ensemble of ZC proxies can substantially increase the performance of model-based predictors.

Similar to the previous experiment, we run both NAS algorithms three different ways: using only the encoding, only the ZC proxies, and both, as features of the predictor. Each algorithm is given 200 architecture evaluations, and we plot performance over time, averaged over 400 trials. See Figure 5 for the results of BANANAS, and Appendix E for the full results. We find that the ZC proxies give the NAS algorithms a boost in performance, especially in the early stages of the search.

6 Conclusions, Limitations, and Broader Impact

In this work, we created `NAS-Bench-Suite-Zero`: an extensible collection of 13 ZC proxies (covering the majority that currently exist), accessible through a unified interface, which can be evaluated on a suite of 28 NAS benchmark tasks. In addition to the codebase, we release precomputed ZC proxy scores across all 13 ZC proxies and 28 tasks, giving 1.5 million total ZC proxy evaluations. This dataset can be used to speed up ZC proxy-based NAS experiments, e.g., from 9 hours to 4

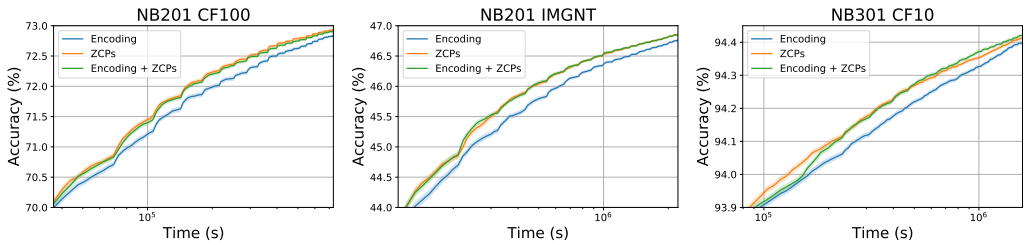

Figure 5: Performance of BANANAS with and without ZC proxies as additional features in the surrogate model. Each curve shows the mean and standard error across 400 trials.

minutes (see Section 3). Overall, `NAS-Bench-Suite-Zero` eliminates the overhead in ZC proxy research, with respect to comparing against different methods and across a diverse set of tasks.

To motivate the usefulness of `NAS-Bench-Suite-Zero`, we conducted a large-scale analysis of the generalizability, bias, and the first information-theoretic analysis of ZC proxies. Our empirical analysis showed substantial complementary information of ZC proxies conditioned on validation accuracy, motivating us to ensemble all 13 into predictor-based NAS algorithms. We show that using several ZC proxies together significantly improves the performance of the surrogate models used in NAS, as well as improving the NAS algorithms themselves.

Limitations and future work. Although our work makes substantial progress towards motivating and increasing the speed of ZC proxy research, there are still some limitations of our analysis. First, our work is limited to empirical analysis. However, we discuss existing theoretical results in Appendix C.1. Furthermore, there are some benchmarks on which we did not give a comprehensive evaluation. For example, on NAS-Bench-301, we only computed ZC proxies on 11 000 architectures, since the full space of 10^{18} architectures is computationally infeasible. In the future, a surrogate model [53, 56] could be trained to predict the performance of ZC proxies on the remaining architectures. Finally, there is very recent work on applying ZC proxies to one-shot NAS methods [52], which tested one ZC proxy at a time with one-shot models. Since our work motivates the ensembling of ZC proxies, an exciting problem for future work is to incorporate 13 ZC proxies into the one-shot framework.

Broader impact. The goal of our work is to make it faster and easier for researchers to run reproducible, generalizable ZC proxy experiments and to motivate further study on exploiting the complementary strengths of ZC proxies. By pre-computing ZC proxies across many benchmarks, researchers can run many trials of NAS experiments cheaply on a CPU, reducing the carbon footprint of the experiments [11, 27]. Due to the notoriously high GPU consumption of prior research in NAS [28, 58], this reduction in CO2 emissions is especially worthwhile. Furthermore, our hope is that our work will have a positive impact in the NAS and automated machine learning communities by showing which ZC proxies are useful in which settings, and showing how to most effectively combine ZC proxies to achieve the best predictive performance. By open-sourcing all of our code and datasets, AutoML researchers can use our library to further test and develop ZC proxies for NAS.

Acknowledgments and Disclosure of Funding

This research was supported by the following sources: Robert Bosch GmbH is acknowledged for financial support; the German Federal Ministry of Education and Research (BMBF, grant RenormalizedFlows 01IS19077C); TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215; the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828; the European Research Council (ERC) Consolidator Grant "Deep Learning 2.0" (grant no. 101045765). Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the ERC. Neither the European Union nor the ERC can be held responsible for them.

Funded by the European Union

References

[1] Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas Donald Lane. Zero-cost proxies for lightweight nas. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[2] Hanlin Chen, Ming Lin, Xiuyu Sun, and Hao Li. Nas-bench-zero: A large scale dataset for understanding zero-shot neural architecture search. *Openreview preprint https://openreview.net/forum?id=hP-SILoczR*, 2021.

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[4] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[5] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16276–16285, 2021.

[6] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[7] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5251–5260, 2021.

[8] Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10480–10490. Curran Associates, Inc., 2020.

[9] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. In *JMLR*, 2019.

[10] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

[11] Karen Hao. Training a single ai model can emit as much carbon as five cars in their lifetimes. *MIT Technology Review*, 2019.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Mojan Javaheripi, Shital Shah, Subhabrata Mukherjee, Tomasz L Religa, Caio CT Mendes, Gustavo H de Rosa, Sebastien Bubeck, Farinaz Koushanfar, and Debadeepta Dey. Litetransformersearch: Training-free on-device search for efficient autoregressive language models. *arXiv preprint arXiv:2203.02094*, 2022.

[14] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[15] Yuhong Li, Cong Hao, Pan Li, Jinjun Xiong, and Deming Chen. Generic neural architecture search via regression. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[16] 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 image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 347–356, 2021.

[17] Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. In *JMLR*, 2020.

[18] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[20] Zhengying Liu, Adrien Pavao, Zhen Xu, Sergio Escalera, Fabio Ferreira, Isabelle Guyon, Sirui Hong, Frank Hutter, Rongrong Ji, Julio C. S. Jacques Junior, Ge Li, Marius Lindauer, Zhipeng Luo, Meysam Madadi, Thomas Nierhoff, Kangning Niu, Chunguang Pan, Danny Stoll, Sebastien Treguer, Jin Wang, Peng Wang, Chenglin Wu, Youcheng Xiong, Arbër Zela, and Yang Zhang. Winning solutions and post-challenge analyses of the chalearn autodl challenge 2019. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3108–3125, 2021.

[21] Vasco Lopes, Saeid Alirezazadeh, and Luís A Alexandre. Epe-nas: Efficient performance estimation without training for neural architecture search. In *International Conference on Artificial Neural Networks*, pages 552–563. Springer, 2021.

[22] 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 *International Conference on Learning Representations*, 2022.

[23] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[24] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[25] Xuefei Ning, Changcheng Tang, Wenshuo Li, Zixuan Zhou, Shuang Liang, Huazhong Yang, and Yu Wang. Evaluating efficient performance estimators of neural architectures. *Advances in Neural Information Processing Systems*, 34, 2021.

[26] Daniel S Park, Jaehoon Lee, Daiyi Peng, Yuan Cao, and Jascha Sohl-Dickstein. Towards nngp-guided neural architecture search. *arXiv preprint arXiv:2011.06006*, 2020.

[27] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[28] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[29] Robin Ru, Clare Lyle, Lisa Schut, Miroslav Fil, Mark van der Wilk, and Yarin Gal. Speedy performance estimation for neural architecture search. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[30] Michael Ruchte, Arber Zela, Julien Siems, Josif Grabocka, and Frank Hutter. Naslib: A modular and flexible neural architecture search library. `https://github.com/automl/NASLib`, 2020.

[31] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[32] David W Scott. Sturges' rule. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(3):303–306, 2009.

[33] Yu Shen, Yang Li, Jian Zheng, Wentao Zhang, Peng Yao, Jixiang Li, Sen Yang, Ji Liu, and Cui Bin. Proxybo: Accelerating neural architecture search via bayesian optimization with zero-cost proxies. *arXiv preprint arXiv:2110.10423*, 2021.

[34] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. *Advances in Neural Information Processing Systems*, 33, 2020.

[35] Yao Shu, Shaofeng Cai, Zhongxiang Dai, Beng Chin Ooi, and Bryan Kian Hsiang Low. Nasi: Label-and data-agnostic neural architecture search at initialization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[36] Yao Shu, Zhongxiang Dai, Zhaoxuan Wu, and Kian Hsiang Low. Unifying and boosting gradient-based training-free neural architecture search. *ArXiv*, abs/2201.09785, 2022.

[37] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[38] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[39] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33:6377–6389, 2020.

[40] Manoel Tenorio and Wei-Tsih Lee. Self organizing neural networks for the identification problem. *Advances in Neural Information Processing Systems*, 1, 1988.

[41] Renbo Tu, Mikhail Khodak, Nicholas Carl Roberts, Nina Balcan, and Ameet Talwalkar. Nas-bench-360: Benchmarking diverse tasks for neural architecture search. *Openreview submission*, 2021.

[42] 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 *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[43] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020.

[44] Chen Wei, Chuang Niu, Yiping Tang, Yue Wang, Haihong Hu, and Jimin Liang. Npenas: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[45] Colin White, Mikhail Khodak, Renbo Tu, Shital Shah, Sébastien Bubeck, and Debadeepta Dey. A deeper look at zero-cost proxies for lightweight nas. In *ICLR Blog Track*, 2022. https://iclr-blog-track.github.io/2022/03/25/zero-cost-proxies/.

[46] Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[47] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[48] Colin White, Arber Zela, Robin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

[49] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

[50] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[51] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9127–9135, 2018.

[52] Lichuan Xiang, Łukasz Dudziak, Mohamed S Abdelfattah, Thomas Chau, Nicholas D Lane, and Hongkai Wen. Zero-cost proxies meet differentiable architecture search. *arXiv preprint arXiv:2106.06799*, 2021.

[53] Shen Yan, Colin White, Yash Savani, and Frank Hutter. Nas-bench-x11 and the power of learning curves. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[54] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[55] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722, 2018.

[56] Arber Zela, Julien Niklas Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate nas benchmarks: Going beyond the limited search spaces of tabular nas benchmarks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[57] Qinqin Zhou, Kekai Sheng, Xiawu Zheng, Ke Li, Xing Sun, Yonghong Tian, Jie Chen, and Rongrong Ji. Training-free transformer architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10894–10903, 2022.

[58] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] [The main claims in the abstract and introduction reflect the paper's contributions and scope.]
   (b) Did you describe the limitations of your work? [Yes] [See Section 6.]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] [See Section 6.]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] [We read the ethics review guidelines and ensured our paper conforms to them.]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] [We did not include theoretical results.]
   (b) Did you include complete proofs of all theoretical results? [N/A] [We did not include theoretical results.]
3. If you ran experiments (e.g. for benchmarks)...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] [We include the code, data, and instructions needed to reproduce the results here: `https://github.com/automl/naslib/tree/zerocost`.]

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] [We discuss all experimental details in Sections 4 and 5, and Appendices D and E.]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] [We report error bars in Sections 4 and 5.]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] [We include the compute and resources used in Section 3.]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] [We cited the creators of all benchmarks we used in Section 2.]

(b) Did you mention the license of the assets? [N/A] [We mention the licenses in Appendix B.]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] [We include a new dataset, available at `https://github.com/automl/naslib/tree/zerocost`.]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] [Our asset does not include data based on people.]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] [Our asset does not include data based on people.]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] [We did not conduct research with human subjects.]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] [We did not conduct research with human subjects.]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] [We did not conduct research with human subjects.]

A NAS Best Practices Checklist

We now describe how we addressed the individual points of the NAS best practice checklist [17].

1. **Best Practices for Releasing Code**

   For all experiments you report:

   (a) Did you release code for the training pipeline used to evaluate the final architectures? [Yes] Since we used NAS benchmarks, we did not evaluate the architectures ourselves. The code for the training pipelines of these benchmarks is publicly available.

   (b) Did you release code for the search space [Yes] Since we used NAS benchmarks, this is already publicly available.

   (c) Did you release the hyperparameters used for the final evaluation pipeline, as well as random seeds? [Yes] Since we used NAS benchmarks, the final evaluation pipeline is fixed. We released our code, including the seeds used.

   (d) Did you release code for your NAS method? [Yes] The code for our NAS method is available at `https://github.com/automl/naslib/tree/zerocost`.

   (e) Did you release hyperparameters for your NAS method, as well as random seeds? [Yes] The hyperparameters used are also available at the above link.

2. **Best practices for comparing NAS methods**

   (a) For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code? [Yes] Since we used NAS benchmarks, the training details are fixed.

   (b) Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)? [Yes] Since we used NAS Benchmarks, these details are fixed automatically.

   (c) Did you run ablation studies? [Yes] We included NAS experiments with only the encoding, only the ZC proxies, and the encoding with 13 ZC proxies.

   (d) Did you use the same evaluation protocol for the methods being compared? [Yes] We used NAS Benchmarks, which keep this fixed.

   (e) Did you compare performance over time? [Yes] Our experiments in Section 5 and Appendix E compare performance over time.

   (f) Did you compare to random search? [No] We used baselines that are better than random search: the original NAS algorithms without ZC proxies.

   (g) Did you perform multiple runs of your experiments and report seeds? [Yes] All of our experiments are averaged across many trials. The seeds are reported in our code files.

   (h) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] All of our experiments use queryable benchmarks.

3. **Best practices for reporting important details**

   (a) Did you report how you tuned hyperparameters, and what time and resources this required? [Yes] We used the default hyperparameters from the respective NAS algorithms and ZC proxies. Our addition of ZC proxies did not add any new hyperparameters.

   (b) Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)? [Yes] Our plots include the end-to-end time.

   (c) Did you report all the details of your experimental setup? [Yes] We included all the details in Section 5 and Appendix E.

B Dataset Documentation

Here, we give an overview of our dataset documentation. For the full details, including links to the dataset, usage, and tutorials, see `https://github.com/automl/NASLib/tree/zerocost`.

Table 5: Licenses for the datasets that we use.

Dataset	License	URL
NAS-Bench-101	Apache 2.0	https://github.com/google-research/nasbench
NAS-Bench-201	MIT	https://github.com/D-X-Y/NAS-Bench-201
NAS-Bench-301	Apache 2.0	https://github.com/automl/nasbench301
TransNAS-Bench-101	MIT	https://github.com/yawen-d/TransNASBench
NAS-Bench-360	MIT	https://github.com/rtu715/NAS-Bench-360

B.1 Author responsibility and license

We, the authors, bear all responsibility in case of violation of rights. The license of our dataset and repository is the **Apache License 2.0**. For more information, see https://github.com/automl/NASLib/blob/Develop/LICENSE.

In addition, we include the licenses of the datasets we used in Table 5.

B.2 Maintenance plan

The data is available on GitHub at https://github.com/automl/NASLib/tree/zerocost. We plan to actively maintain the repository, and we also welcome contributions from the community. For more information, see https://github.com/automl/NASLib/tree/zerocost.

B.3 Code of conduct

Our Code of Conduct is from the Contributor Covenant, version 2.0. See https://www.contributor-covenant.org/version/2/0/code_of_conduct.html. The policy is copied below.

> "We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation."

B.4 Datasheet

We include a datasheet [10] for NAS-Bench-Suite-Zero.

Motivation For Datasheet Creation

*Why was the datasheet created? (e.g., was there a specific task in mind? was there a specific gap that needed to be filled?) The goal of our work is to make it easier and faster for researchers to run generalizable, reproducible ZC proxy experiments, and to motivate further study on exploiting the complementary strengths of ZC proxies. By pre-computing ZC proxies across many benchmarks, users can run many trials of NAS experiments cheaply on a CPU, reducing their carbon footprint [11, 27]. Since prior research in NAS has notoriously high GPU consumption [28, 58], this reduction in CO2 emissions is worthwhile.

*Has the dataset been used already? If so, where are the results so others can compare (e.g., links to published papers)? The dataset has only been used in this paper. See Sections 4 and 5 and Appendix D and E.

*What (other) tasks could the dataset be used for? Since the dataset only contains values of ZC proxies on existing NAS benchmarks, we are not aware of any tasks this dataset can be used for, besides analyzing ZC proxies and speeding up ZC proxy-based NAS algorithms.

*Who funded the creation dataset? This dataset was created by researchers at the University of Freiburg, Abacus.AI, the University of Toronto, and the Bosch Center for Artificial Intelligence. Funding for the dataset computation itself is from the University of Freiburg.

*Any other comment? None.

Datasheet Composition

*What are the instances?(that is, examples; e.g., documents, images, people, countries) Are there multiple types of instances? (e.g., movies, users, ratings; people, interactions between them; nodes, edges) For each NAS benchmark, each instance is a tuple of an architecture hash, the name of a ZC proxy, and the value and runtime of the ZC proxy evaluated on that architecture.

*How many instances are there in total (of each type, if appropriate)? See Table 2 for a full breakdown of the number of instances for each NAS benchmark.

*What data does each instance consist of ? "Raw" data (e.g., unprocessed text or images)? Features/attributes? Is there a label/target associated with instances? If the instances related to people, are subpopulations identified (e.g., by age, gender, etc.) and what is their distribution? Each instance is a tuple of an architecture hash, the name of a ZC proxy, and the value and runtime of the ZC proxy evaluated on that architecture. These will most-often be used to speed up NAS experiments or run analysis on ZC proxies, in which case they are not used as features/labels.

*Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text. There is no missing information from individual instances.

*Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)? If so, please describe how these relationships are made explicit. There are no relationships between individual instances.

*Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set? If the dataset is a sample, then what is the larger set? Is the sample representative of the larger set (e.g., geographic coverage)? If so, please describe how this representativeness was validated/verified. If it is not representative of the larger set, please describe why not (e.g., to cover a more diverse range of instances, because instances were withheld or unavailable). NAS-Bench-201 and TransNAS-Bench-101-Micro and Macro contain all possible instances. NAS-Bench-101, NAS-Bench-301, and the additional architectures evaluated on spherical-cifar, SVHN, and NinaPro are samples. All samples are drawn uniformly at random from the respective search space. This is ensured because the code used to draw architectures uniformly at random is from the respective original repositories that introduced the NAS benchmarks.

*Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them. The main usage of this dataset is to speed up NAS experiments, for which there are no data splits. For experiments involving architecture prediction (such as the standalone predictor experiments in Section 5, we do not give recommended data splits but instead recommended running at least 100 trials, where each trial randomly samples train and (disjoint) test sets.

*Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description. There are no known errors, sources of noise, or redundancies.

*Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? If it links to or relies on external resources, a) are there guarantees that they will exist, and remain constant, over time; b) are there official archival versions of the complete dataset (i.e., including the external resources as they existed at the time the dataset was created); c) are there any restrictions (e.g., licenses, fees) associated with any of the external resources that might apply to a future user? Please provide descriptions of all external resources and any restrictions associated with them, as well as links or other access points, as appropriate. The dataset does rely on the code from the respective existing NAS benchmarks to reconstruct the architecture itself from the hash provided in our dataset. Furthermore, a user will often want access to the validation accuracies of the architectures in our dataset, which also comes from the existing NAS benchmarks. Since these NAS benchmarks serve similar goals as our dataset (to accelerate and simplify research in NAS) and are hosted similarly to ours (on Google Drive and GitHub), we are confident that these benchmarks will exist and remain constant over time. In some cases, we have also created our own versions of the NAS benchmarks, so all of the data can be downloaded at one time. Licenses and links are described in Table 5.

Any other comments? None.

Collection Process

*What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)? How were these mechanisms or procedures validated? The data was created with a software program (available at `https://github.com/automl/NASLib/tree/zerocost`). The ZC proxy code were taken from their original repositories. All ZC proxies from Table 1 were run on an Intel Xeon Gold 6242 CPU, using a batch size of 64, except for the case of TransNAS-Bench-101: due to the extreme memory usage of the Taskonomy tasks ($>$ 30GB memory), we used a batch size of 32.

*How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe how. As described, all data was created with a publicly available software program.

*If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)? As described earlier, the sampling was done uniformly at random.

*Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)? The data collection process (e.g., running the code) was done by the authors of this work.

*Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created. The total computation time for all 1.5M evaluations was 1100 CPU hours on Intel Xeon Gold 6242 CPUs (using up to 20 CPUs and 150 cores in parallel). The timeframe was May 15, 2022 to June 1, 2022.

Data Preprocessing

*Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remainder of the questions in this section. There was no preprocessing that needed to be done.

*Does this dataset collection/processing procedure achieve the motivation for creating the dataset stated in the first section of this datasheet? If not, what are the limitations? Yes, the dataset collection procedure achieves our motivation. See Table 8 for a list of the speedups in NAS experiments achieved when using our dataset.

*Any other comments None.

Dataset Distribution

*How will the dataset be distributed? (e.g., tarball on website, API, GitHub; does the data have a DOI and is it archived redundantly?) The dataset is on Google Drive, with a DOI.

*When will the dataset be released/first distributed? What license (if any) is it distributed under? The dataset is public as of June 8, 2022, distributed under the Apache License 2.0.

*Are there any copyrights on the data? There are no copyrights on the data.

*Are there any fees or access/export restrictions? There are no fees or restrictions.

*Any other comments? None.

Dataset Maintenance

*Who is supporting/hosting/maintaining the dataset? The authors of this work are supporting/hosting/maintaining the dataset.

*Will the dataset be updated? If so, how often and by whom? If new NAS benchmarks are created in the NAS research community, the authors of this work may update `NAS-Bench-Suite-Zero` to include ZC proxy values for the new benchmarks. Similarly, if new ZC proxies are relased, the authors may update `NAS-Bench-Suite-Zero` to include the new ZC proxies.

*How will updates be communicated? (e.g., mailing list, GitHub) Updates will be communicated on the GitHub README of this project.

*If the dataset becomes obsolete how will this be communicated? If the dataset becomes obsolete, it will be communicated on the GitHub README of this project.

*If others want to extend/augment/build on this dataset, is there a mechanism for them to do so? If so, is there a process for tracking/assessing the quality of those contributions. What is the process for communicating/distributing these contributions to users? Others can create a pull request or raise an issue on GitHub with possible extensions/augmentations to our dataset, which will be approved in a case-by-case basis. For example, an author of a new ZC proxy may create a PR in our codebase with the new ZC proxy, and then we will evaluate the ZC proxy on all architectures in `NAS-Bench-Suite-Zero` and update the dataset. These updates will again be communicated on the GitHub README.

Legal and Ethical Considerations

*Were any ethical review processes conducted (e.g., by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation. There was no ethical review process. We note that our dataset was created by simply by running ZC proxy computations on architectures of existing NAS benchmarks, in some cases using publicly available, licensed datasets such as CIFAR-10 or CIFAR-100.

*Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctorpatient confidentiality, data that includes the content of individuals non-public communications)? If so, please provide a description. The dataset does not contain any confidential data.

*Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why None of the data might be offensive, insulting, threatening, or otherwise cause anxiety.

*Does the dataset relate to people? If not, you may skip the remaining questions in this section. The dataset does not relate to people.

*Any other comments? None.

C Related Work Continued

In this section, we give additional details on related work, continued from Section 2.

Multiple recent works have investigated the performance of ZC proxies in ranking architectures over different NAS benchmarks. [25] provides rank correlations and pairwise correlations of 10 ZC proxies across 7 tasks, and concludes that the relative performance of different ZC proxies highly depends on the search space. They further analyze how ZC proxies have improper biases. [48] compares 6 ZC proxies across four tasks, and further shows how `jacov` can be used to accelerate the search in predictor-based NAS. In particular, OMNI [48] combines `jacov` with *sum of training losses* [29] in the surrogate models of BANANAS and predictor-guided evolution. However, the predictor-based NAS experiments are restricted to NAS-Bench-201 and a single ZC proxy. Similar to [48], ProxyBO [33] introduces a NAS framework based on BO which uses ZC proxies to speed up NAS. It dynamically chooses whether to use a Gaussian process, `snip`, `jacov`, or `synflow` as the surrogate model in BO. Experiments were done on five tasks. Note that although the NAS method makes use of three different ZC proxies, each are used *separately* to make predictions on the performance of architectures.

Recently, NAS-Bench-Zero was introduced [2], a new benchmark based on popular computer vision models ResNet [12] and MobileNetV2 [31], and examined different characteristics of 10 ZC proxies across these search space as well as three existing search spaces. The study shows in particular that individual ZC proxies do not transfer across NAS benchmarks. They also show that voting among `synflow`, `zen`, `snip` and `synflow` is the optimal voting ZC proxy strategy. A recent overview of ZC proxies [45] computes rank correlation, pairwise correlation, and performance plots for 8 ZC proxies across 12 tasks.

Only two prior works combine the information of multiple ZC proxies together in architecture predictions [1, 2] and both only use the *voting* strategy to combine three or four ZC proxies. Our work is the first to combine ZC proxies in a nontrivial way, and the first to combine 13 ZC proxies. We also conduct analysis on the largest set of ZC proxies and benchmarks to date.

C.1 Theoretical results for ZC proxies

While ZC proxies are starting to be used more widely today [1, 13, 45, 57], still relatively little is known about them from a theoretical standpoint. However, there have been a few works that do give theoretical results. In this section, we survey the existing theoretical results for ZC proxies.

Ning et al. gave a theoretical preference analysis for `synflow`, proving that it favors larger architectures (Section B.3 in [25]). Specifically, they prove that given an architecture, introducing a new fully-connected layer into an MLP architecture causes the `synflow` value to increase. The core of their argument is to prove the following statement: "when introducing a new fully-connected layer, the expected loss gradients with respect to the existing parameters increases." The authors also claim that the intuition for this argument should extend to convolutional neural networks. Finally, we note that our empirical results from Table 3 confirm their theoretical finding.

Shu et al. [36] attempted to give a unified, general theory for multiple ZC proxies. First, the authors prove that ZC proxy values are asymptotically similar. Specifically, they show that assuming the loss function of the neural network is β-Lipschitz continuous, and γ-Lipschitz smooth, then with high-priority, then the values of `grad_norm`, `snip`, and `grasp` are all asymptotically similar up to constants (i.e., the same under big-Oh notation) to the trace norm of the NTK matrix at initialization. This result implies that the values of these ZC proxies are highly correlated.

Next, Shu et al. establish generalization bounds for DNNs in terms of the ZC proxies. Specifically, they show that the generalization error of a DNN is at most the sum of the training error of the DNN and $O\left(\kappa/\mathcal{M}\right)$, where $\mathcal{M}$ can be set to `grad_norm`, `snip`, or `grasp`, and κ is the condition number of the NTK matrix at initialization, i.e., given the NTK matrix Θ_0, $\kappa = \lambda_{\max}(\Theta_0)/\lambda_{\min}(\Theta_0)$.

As a corollary, they also bound the generalization error in terms of the ZC proxy value and other fixed constants of the neural network, without the training error term.

Other than these results, a few works have derived new ZC proxies via a theoretical analysis or inspired by existing theories of deep learning. Shu et al. [35] introduce NASI by giving a theoretical analysis that shows the trace norm of the NTK has a similar form to gradient flow. Other theory-inspired ZC proxies include TE-NAS [4], which uses the spectrum of the NTK and the number of linear regions in the input space, and NNGP-NAS [26], which approximates the Neural Network Gaussian Process using Monte-Carlo methods.

Table 6: Spearman rank correlation for 100 architectures randomly drawn from the FBNet search space on various ZC proxies.

ZC Proxy	fisher	flops	grad_norm	grasp	jacov	params	snip	synflow
Spearman	0.2574	**0.6484**	0.4278	-0.262	-0.0895	0.3762	0.5102	0.4954

As ZC proxies gain in popularity, a further theoretical analysis is an important step in understanding their robustness on different datasets, and in designing higher-performing ZC proxies.

D Details from Section 4

In this section, we give additional details from Section 4.

D.1 Details from Section 4.1: generalization

We give the full extensions of the experiments from Section 4.1. In Figure 6, for each ZC proxy and each benchmark, we compute the Spearman rank correlation (see Section 4). This is the full version of Figure 2.

In Figure 7, we compute the Pearson correlation coefficient between ZC proxy scores on pairs of benchmarks. This is the full version of Figure 3.

Next, we recompute Figure 2 using different metrics: Precision@K and BestRanking@K [2, 25]. Let M denote the number of architectures, and for each architecture a_i from $i \in [1, M]$, denote the rankings of the ground truth and ZC proxy-estimated scores are r_i and n_i, respectively. Given K, define $A_K = \{a_i \mid n_i < KM\}$. The definitions are as follows:

$$\text{Precision@}K = \frac{\#\{i \mid r_i < K \wedge n_i < K\}}{K}$$
$$\text{BestRanking@}K = \text{argmin}_{\alpha_i \in A_K} r_i/M$$

In Figure 8, we recompute Figure 2 using Precision@K, for $K = 5, 25, 100$. In Figure 9, we recompute Figure 2 using BestRanking@K, for $K = 5, 25, 100$. Overall, we see similar trends to Figure 2, but we note that Precision@K and BestRanking@K may be more useful than Spearman in terms of NAS, since the goal of NAS is to find the very best architectures.

D.1.1 Initial results with FBNet

While `NAS-Bench-Suite-Zero` contains 28 tasks, the majority of search spaces used were designed for research. Now, in contrast, we give initial results for FBNet [50] as a search space that has been used to achieve state-of-the-art results.

The FBNet search space consists of 22 searchable layers, with 9 operation choices each (3 filters and 3 kernel sizes), for a total of $9^{22} = 10^{21}$ architectures in the search space. The block structure is inspired by MobileNetV2 [31] and ShiftNet [51].

See Table 6 for the Spearman rank correlation values of the validation accuracy of 100 randomly drawn architectures compared to ZC proxies. Even though the FBNet search space is size 10^{21}, some of the ZC proxies perform surprisingly well, such as `snip`, `synflow`, and `flops`. The highest-performing ZC proxy is `flops`.

D.2 Details from Section 4.2: information theory

In this section, we give details from Section 4.2. We start with more details on the conditionial entropy, including why we chose this metric, how it is computed, and how to interpret the results.

- *Why do we choose conditional entropy as the metric?*
  The conditional entropy of a random variable Y given another random variable X is

$$H(Y|X) = \mathbb{E}[-\log(p(y|x))] = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)}, \tag{3}$$

  for two support sets $\mathcal{X}, \mathcal{Y}$. If we assume entropy to be a measure of information, in other words uncertainty within a random variable, conditional entropy essentially captures what is left of the uncertainty after conditioning. $H(Y|X)$ also has certain desirable properties: (1). $H(Y|X) = 0$ if and only if X completely determines the value of Y; (2). $H(Y|X) = H(Y)$ if and only if X and Y are completely independent; and (3). $H(Y|X_1, X_2) = H(Y, X_1, X_2) - H(X_1, X_2)$. We can then easily calculate conditional entropy when conditioning on multiple random variables, and use it as a metric for uncertain information.
- *Discretization of ZC proxy scores and ground-truth accuracies.*
  Calculating conditional entropy as prescribed above requires that all random variables be discrete, which is not the case for raw validation accuracies and ZC proxy scores. Implementation wise, we discretize all the float values and use Sturge's rule [32] as a heuristic to choose the number of bins for discretization:

$$n_{\text{bins}} = \text{round}(1 + 3.322 * \log(N))), \text{where } N \text{ is the sample size.} \tag{4}$$

  Therefore, information about Y does not reveal the exact validation accuracy but rather the interval in which the value falls.
- *Interpreting the information gain heatmap.*
  The information gain heatmap shows how much the conditional entropy of $y|z_{i_1}$ decreases to $y|z_{i_1}, z_{i_2}$ as the scores of ZC proxy on each column (z_{i_2}) is revealed, given that we already know the scores of ZC proxy on each row (z_{i_1}). For instance, on Figure 4 (top right), the value 1.42 on the second row, first column shows that $H(y|scores(\text{synflow}) - H(y|scores(\text{synflow}), scores(\text{epe\_nas})) = 1.42$. Note that (1). all values on the diagonal are 0.0 because no information is gained when we add a copy of the existing ZC proxy scores; (2). The heatmap is **not** symmetric like pairwise conditional entropy. The order in which conditioning is applied affects the amount of information gain, i.e. $\textbf{IG}(y|z_{i_1}, z_{i_2}) \neq \textbf{IG}(y|z_{i_2}, z_{i_1})$; (3). **IG** measures how much one ZC proxy's information complements that of another for determining the ground-truth accuracy. It does **not** serve as a direct indicator of the quality of individual ZC proxy themselves.
- *Interpreting the entropy vs. number of ZC proxies plot.*
  Conditional entropy monotonically decreases as we condition the validation accuracy, y, on an increasing amount of ZC proxy scores, $z_{i_1}, \ldots z_{i_k}$, which always brings in additional information. In most cases, marginal **IG** drastically decreases as the amount of ZC proxies k reaches 4, but this is only true if the proxies are chosen strategically, using either a greedy or a brute-force minimization approach. For the majority of benchmarks, the less computationally intensive greedy strategy matches up to the brute-force strategy. On the other hand, randomly choosing the ZC proxies does not have stable performance and could be suboptimal, such as on NAS-Bench-201 + CIFAR-100 in Figure 4 (bottom middle).

For completion, in Figure 10, we plot the average pairwise correlation for all pairs of ZC proxies.

In Figures 11, 12, 13, 14, 15, we show all the conditional entropy and information gain heatmaps, in addition to the entropy vs. number of ZC proxies plots for all benchmark, dataset pairs. Note that for TransNAS-Bench-101, there are no results for `epe_nas` because it is not defined on non-classification tasks. Similarly, `synflow` returns 0.0 for certain non-classification tasks such as the ones in TransNAS-Bench-101, so we also removed `synflow` from the TransNAS-Bench-101 plots.

While the conditional entropy and information gain plots from Figure 4 was computed using Equation 4 to compute the number of bins, we also run the same experiment using a different discretization strategy: the bin dividers are computed based on percentages of the data. See Figure 16 (top). While the scales differ, we see largely the same trends. For example, there is still a cluster among `nwot`, `flops`, `l2_norm`, `zen`, and `params`. This suggests that this analysis is robust to the two different discretization strategies. Next, we also re-run the experiment on conditional entropy vs. k from Figure 4 using the top 1000 architectures only, which may be important in the context of NAS, since NAS is concerned with finding the *best* architectures. See Figure 16 (bottom). We find that the random ordering performs comparatively better, predictably implying that it is harder to distinguish architectures that are in the top 1000 vs. randomly drawn architectures.

Table 7: Pearson correlation coefficients between predictors and bias metrics (in bold) on different datasets, for the most and least biased ZC proxies on each search space and task. For example, for the **Conv:pool** bias on NB201-CF10, `synflow` is most biased, with a correlation of 0.76, while `grasp` is least biased (in terms of absolute value), with a correlation of -0.01.

Name	Conv:pool		Cell size		Num. skip connections		Num. parameters	
	Most biased	Least biased	Most biased	Least biased	Most biased	Least Biased	Most biased	Least biased
NB101-CF10	synflow 0.76	grasp -0.01	n/a	n/a	n/a	n/a	nwot 0.74	epe_nas -0.02
NB201-CF10	l2_norm 0.87	grasp 0.01	synflow 0.57	grasp -0.02	l2_norm -0.41	grasp -0.01	l2_norm 0.70	grasp 0.00
NB201-CF100	l2_norm 0.87	grasp 0.01	synflow 0.57	grasp -0.01	l2_norm -0.41	grasp -0.01	l2_norm 0.70	fisher 0.01
NB201-IM	l2_norm 0.87	grasp 0.01	synflow 0.58	grasp 0.01	l2_norm -0.41	grasp -0.01	l2_norm 0.70	grasp 0.01
NB301-CF10	params 0.78	fisher 0.01	n/a	n/a	flops -0.35	epe_nas 0.01	zen 0.99	epe_nas -0.01
TNB101_MICRO-JIGSAW	n/a	n/a	l2_norm 0.70	grasp -0.02	plain 0.50	grasp -0.01	l2_norm 0.64	grasp 0.02
TNB101_MICRO-SCENE	n/a	n/a	l2_norm 0.70	fisher 0.07	plain 0.49	grasp -0.10	snip 0.64	grasp -0.04
TNB101_MICRO-OBJECT	n/a	n/a	l2_norm 0.70	fisher -0.08	plain 0.49	grasp -0.06	l2_norm 0.64	grasp -0.02
TNB101_MICRO-AUTOENC	n/a	n/a	l2_norm 0.70	grasp -0.02	grad_norm -0.46	grasp -0.03	l2_norm 0.64	grasp 0.02
TNB101_MICRO-NORMAL	n/a	n/a	l2_norm 0.70	plain 0.01	snip -0.45	grasp -0.01	l2_norm 0.64	plain 0.00
TNB101_MICRO-ROOM	n/a	n/a	l2_norm 0.70	fisher 0.10	plain 0.45	jacov 0.14	l2_norm 0.64	grasp -0.01
TNB101_MICRO-SEGMENT	n/a	n/a	l2_norm 0.70	grasp 0.00	grad_norm -0.43	grasp 0.01	l2_norm 0.64	grasp -0.01
TNB101_MACRO-JIGSAW	n/a	n/a	n/a	n/a	n/a	n/a	l2_norm 0.89	plain 0.04
TNB101_MACRO-SCENE	n/a	n/a	n/a	n/a	n/a	n/a	l2_norm 0.90	plain 0.05
TNB101_MACRO-OBJECT	n/a	n/a	n/a	n/a	n/a	n/a	l2_norm 0.89	plain 0.05
TNB101_MACRO-AUTOENC	n/a	n/a	n/a	n/a	n/a	n/a	l2_norm 0.89	plain 0.01
TNB101_MACRO-NORMAL	n/a	n/a	n/a	n/a	n/a	n/a	l2_norm 0.89	grasp -0.02
TNB101_MACRO-ROOM	n/a	n/a	n/a	n/a	n/a	n/a	l2_norm 0.89	grasp 0.00
TNB10_MACRO-SEGMENT	n/a	n/a	n/a	n/a	n/a	n/a	l2_norm 0.89	plain 0.00

D.3 Details from Section 4.3: biases

In this section, we give details from Section 4.3. In Table 7, for each bias metric we assess, we show the ZC proxies with the highest and lowest absolute correlation for each search space and dataset, if applicable. For the number of parameters bias, we do not consider the ZC proxies of `params` and `flops` since they trivially have 1.00 correlation. Note that operation biases are not available in TransNASBench101-Macro because the search space is architecture-level. This is an extension of Table 3.

D.4 NAS-Bench-Suite-Zero Speedup Details

Here we show statistics on how our benchmark speeds up NAS experiments previously done with NAS-Bench-Suite by orders of magnitude. See Table 8.

Table 8: Runtimes (on an Intel Xeon Gold 6242 CPU) for all types of experiments done in this paper, with and without `NAS-Bench-Suite-Zero`. The runtimes of the experiments with NBSuite are computed by using the average training times for randomly drawn architectures from each search space in NBSuite.

Experiment	With NBSuite (approx.)	With NBSuite + NBSuite-Zero	Speedup
Mutual information study	158.2 hours	124.1 seconds	4592×
Architecture bias study	6956 hours	14.8 seconds	1776003×
Standalone XGBoost+ZC, 100 trials	1033 hours	100 seconds	37180×
BANANAS+ZC, 100 trials	4694 hours	4260 seconds	3967×
NPENAS+ZC, 100 trials	1033 hours	3470 seconds	1071×

Table 9: Average Spearman rank correlations between XGBoost predictions and validation accuracies, for each benchmark, across three different experiments: *Encoding* uses only the encoding of the model, *ZC* uses only the ZC features, and *Both* concatenates ZC features to the encoding of the model. 100 models were used to train XGBoost.

Features Benchmark	Encoding	ZC	Both	% Improvement (ZC)	% Improvement (Both)
NB101-CF10	0.546	0.708	0.718	29.67	31.50
NB201-CF10	0.622	0.905	0.906	45.50	45.66
NB201-CF100	0.640	0.907	0.908	41.71	41.87
NB201-IMGNT	0.683	0.879	0.883	28.70	29.28
NB301-CF10	0.314	0.405	0.465	28.98	48.09
TNB101_MACRO-AUTOENC	0.673	0.831	0.837	23.48	24.37
TNB101_MACRO-JIGSAW	0.809	0.706	0.809	-12.73	0.00
TNB101_MACRO-NORMAL	0.617	0.710	0.716	15.07	16.05
TNB101_MACRO-OBJECT	0.736	0.840	0.843	14.13	14.54
TNB101_MACRO-ROOM	0.683	0.589	0.707	-13.76	3.51
TNB101_MACRO-SCENE	0.832	0.891	0.899	7.09	8.05
TNB101_MACRO-SEGMENT	0.900	0.807	0.876	-10.33	-2.67
TNB101_MICRO-AUTOENC	0.714	0.754	0.803	5.60	12.46
TNB101_MICRO-JIGSAW	0.585	0.730	0.743	24.79	27.01
TNB101_MICRO-NORMAL	0.657	0.801	0.809	21.92	23.14
TNB101_MICRO-OBJECT	0.637	0.733	0.752	15.07	18.05
TNB101_MICRO-ROOM	0.582	0.843	0.844	44.85	45.02
TNB101_MICRO-SCENE	0.710	0.849	0.866	19.58	21.97
TNB101_MICRO-SEGMENT	0.767	0.886	0.897	15.51	16.95

E Details from Section 5

In this section, we give the full details from Section 5.

We start by presenting the complete standalone predictor experiments. In Section 5, we mentioned that on NAS-Bench-201 CIFAR-100, the Spearman rank correlation of XGBoost predictions trained on 100 randomly sampled architectures and averaged over 100 trials, improves from 0.640 to 0.908 when 13 ZC proxies are added. Now, we present the results of this same experiment for all benchmarks. See Table 9. We see that the large improvement is consistent across the board. We also run the same experiment when XGBoost is trained on 1000 randomly sampled architectures. See Table 10. Even though the predictions with the original XGBoost already have high rank correlation, we show that ZC proxies improve the performance even more.

E.1 Feature importances of ZC proxies

In this section, we train an XGBoost surrogate model on 100 and 1000 randomly drawn architectures using the ZC proxies as features, and then we plot feature importances for each feature. The feature importance is calculated by the the number of times a feature is used to split the data across all trees (the default feature importance method in the XGBoost library [3]). See Figures 20 and 21 for the results with a training set size of 100 and 1000, respectively.

Table 10: Average Spearman rank correlations between XGBoost predictions and validation accuracies, for each benchmark, across three different experiments: *Encoding* uses only the encoding of the model, *ZC* uses only the ZC features, and *Both* concatenates ZC features to the encoding of the model. 1000 models were used to train XGBoost.

Features Benchmark	Encoding	ZC	Both	% Improvement (ZC)	% Improvement (Both)
NB101-CF10	0.748	0.811	0.851	8.42	13.77
NB201-CF10	0.890	0.954	0.961	7.19	7.98
NB201-CF100	0.906	0.953	0.959	5.19	5.85
NB201-IMGNT	0.922	0.948	0.957	2.82	3.80
NB301-CF10	0.678	0.496	0.705	-26.84	3.98
TNB101_MACRO-AUTOENC	0.890	0.903	0.917	1.46	3.03
TNB101_MACRO-JIGSAW	0.812	0.801	0.856	-1.35	5.42
TNB101_MACRO-NORMAL	0.692	0.759	0.764	9.68	10.40
TNB101_MACRO-OBJECT	0.846	0.880	0.888	4.02	4.96
TNB101_MACRO-ROOM	0.741	0.731	0.793	-1.35	7.02
TNB101_MACRO-SCENE	0.936	0.936	0.953	0.00	1.82
TNB101_MACRO-SEGMENT	0.951	0.920	0.952	-3.26	0.11
TNB101_MICRO-AUTOENC	0.838	0.815	0.861	-2.74	2.74
TNB101_MICRO-JIGSAW	0.768	0.827	0.833	7.68	8.46
TNB101_MICRO-NORMAL	0.816	0.850	0.864	4.17	5.88
TNB101_MICRO-OBJECT	0.806	0.841	0.858	4.34	6.45
TNB101_MICRO-ROOM	0.874	0.943	0.947	7.89	8.35
TNB101_MICRO-SCENE	0.862	0.929	0.943	7.77	9.40
TNB101_MICRO-SEGMENT	0.921	0.934	0.948	1.41	2.93

E.2 Ablation study on the number of ZC proxies

Next, we give an ablation study on the number of ZC proxies as features, for an XGBoost surrogate model trained on 1000 randomly drawn architectures. The ordering of ZC proxies is computed via the greedy method from Section 4.3. See Figure 17. We find that on all tasks, the best performance is achieved with all 13 ZC proxies (in some cases, there are ties). However, after 6-8 ZC proxies, there is only a small improvement up to the full 13 ZC proxies. This is consistent with our mutual information study from Section 4.3.

E.3 Additional NAS results

Finally, we present more NAS results, extending the NAS results from Section 5. In Figure 18, we run BANANAS in the same setting as Section 5, on 11 benchmarks. We see that ZC proxies improve performance across the board. In Figure 19, we run the same experiment with NPENAS instead of BANANAS. Note that since NPENAS requires a mutation step, we are only able to run it on complete benchmarks: NAS-Bench-201 and TransNAS-Bench-101 (in particular, not NAS-Bench-101 or NAS-Bench-301).

F ZC Proxy Competition

`NAS-Bench-Suite-Zero` was used successfully in the Zero Cost NAS Competition at AutoML-Conf 2022. During the competition, participants developed new, better versions of ZC proxies in the `NAS-Bench-Suite-Zero` codebase. The challenge was as follows: given N models, the participant's ZC proxy will be used to rank the models for a specified task. The Kendall-Tau rank correlation is used to score the metric, averaged across three benchmarks in the test phase of the competition. The tasks in the development phase of the competition were NB201 with Ninapro and SVHN, NB301 with Ninapro and SVHN, and TNB101-Micro with Ninapro, SVHN, and Spherical-CIFAR100. The tasks in the final test phase of the competition were NB101 with CIFAR10, NB201 with ImageNet16x120, NB301 with CIFAR10, TNB101-Macro with Object Classification, and TNB101-Micro with Object Classification. The winning teams used a normalized version of `synflow`, a normalized version of `fisher`, and a product of `grad_norm` and `params`. For more information, see the competition homepage. [3]

[3] See `https://sites.google.com/view/zero-cost-nas-competition/home`.

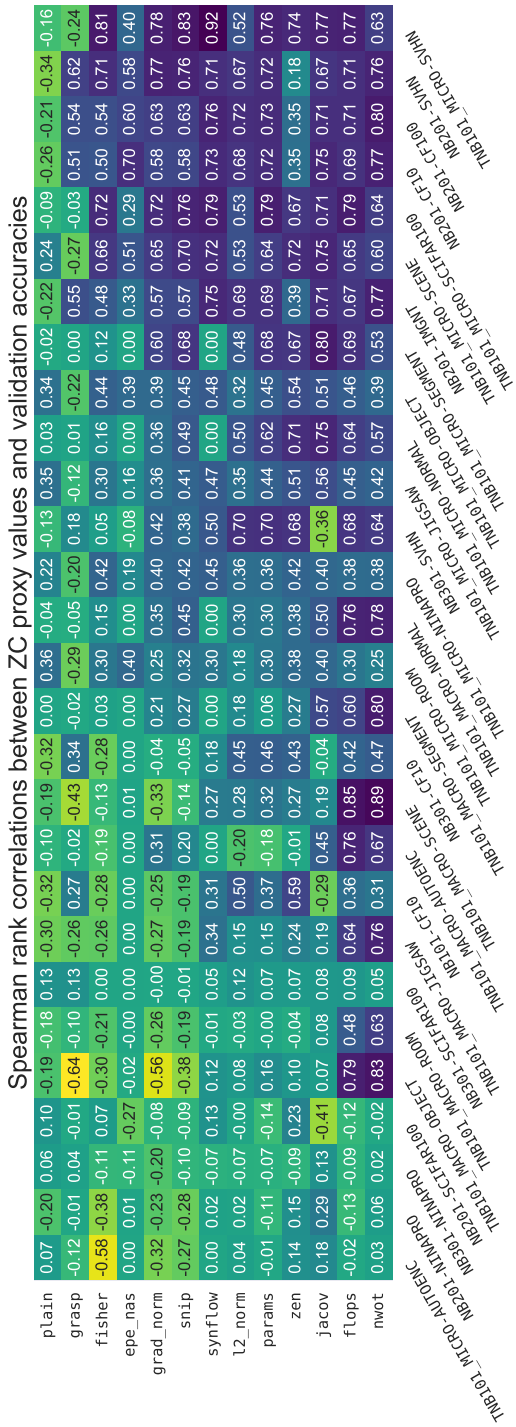

Figure 6: Spearman rank correlation coefficient between ZC proxy values and validation accuracies, for each ZC proxy and benchmark. The rows and columns are ordered based on the mean scores across columns and rows, respectively. This is the full version of Figure 2.

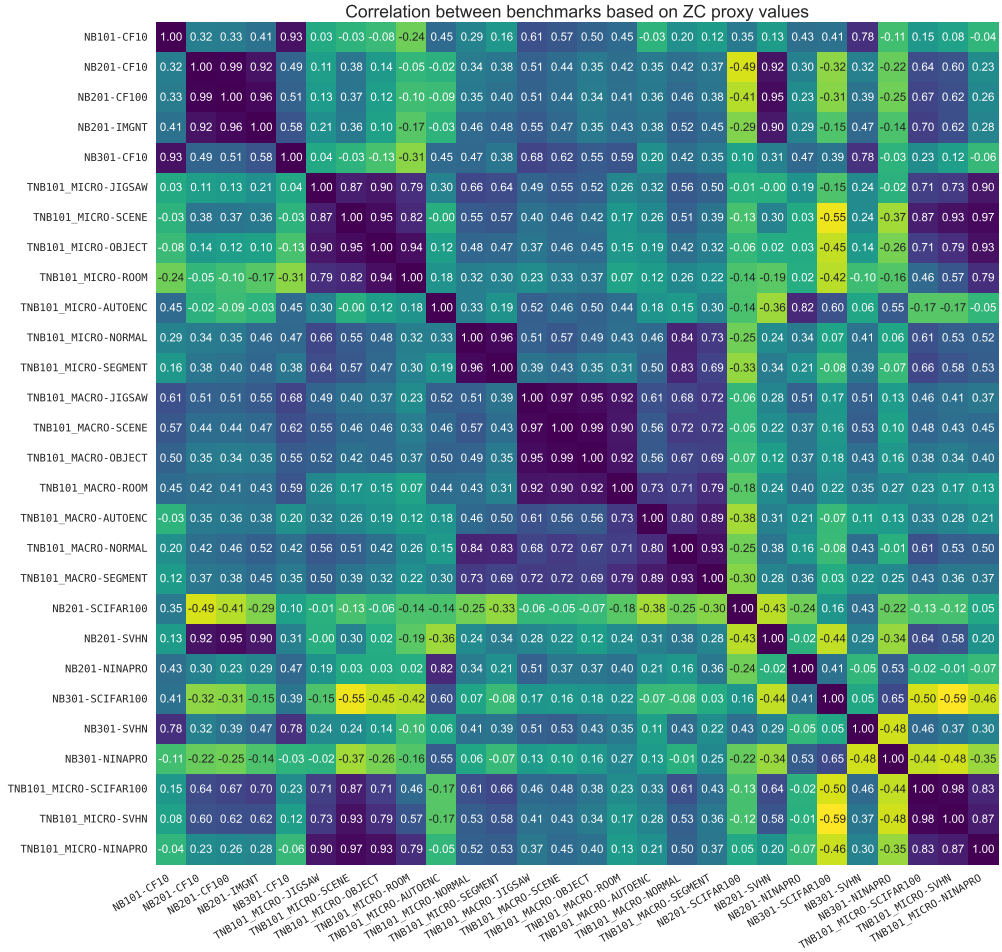

Figure 7: Pearson correlation coefficient between ZC proxy scores on pairs of benchmarks. The entries in the plot are ordered based on the mean score across each row and column. This is the full version of Figure 3.

Figure 8: Precision@K between ZC proxy values and validation accuracies, for each ZC proxy and benchmark. The rows and columns are ordered based on the mean scores across columns and rows, respectively.

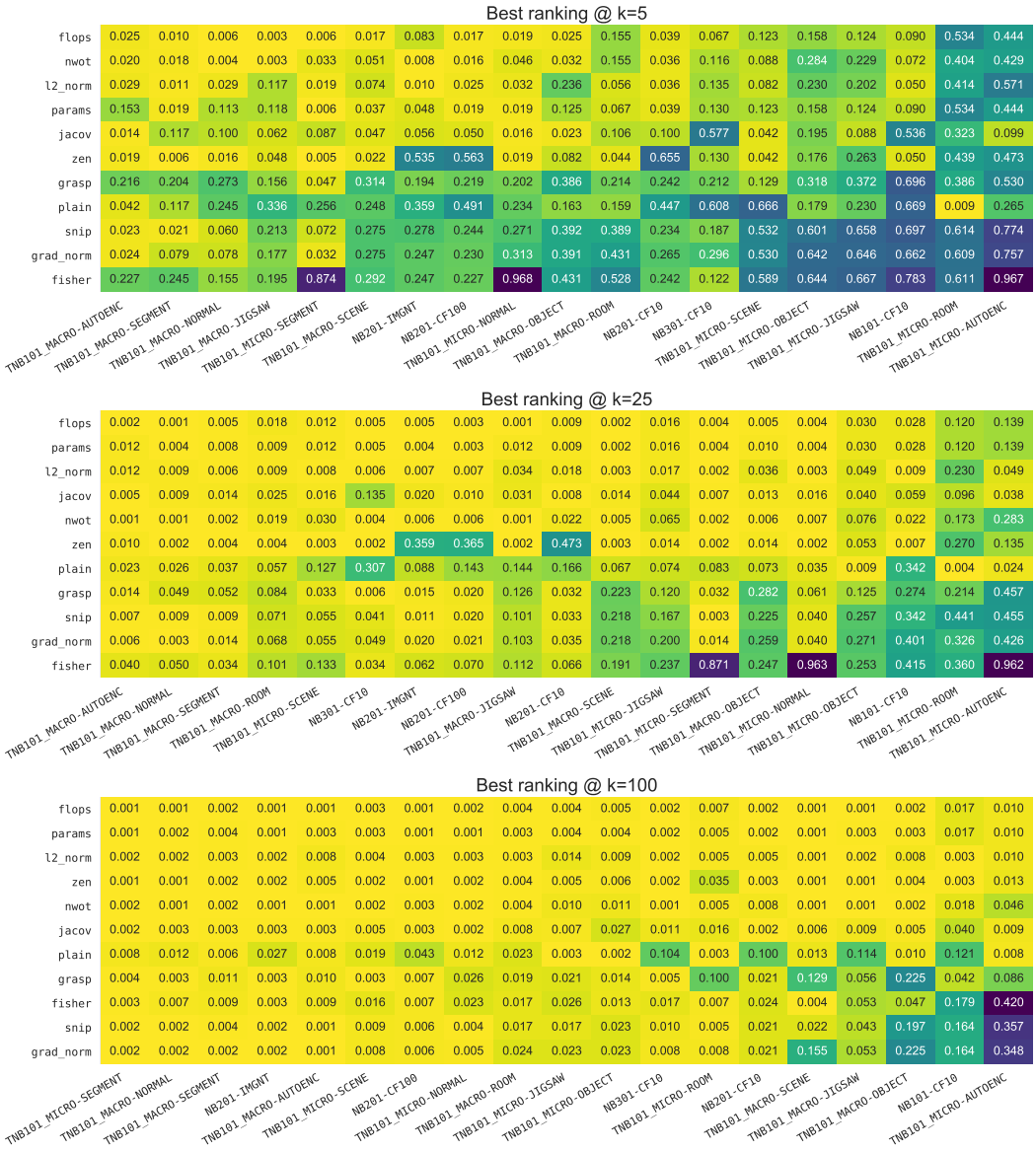

Figure 9: BestRanking@K between ZC proxy values and validation accuracies, for each ZC proxy and benchmark. The rows and columns are ordered based on the mean scores across columns and rows, respectively.

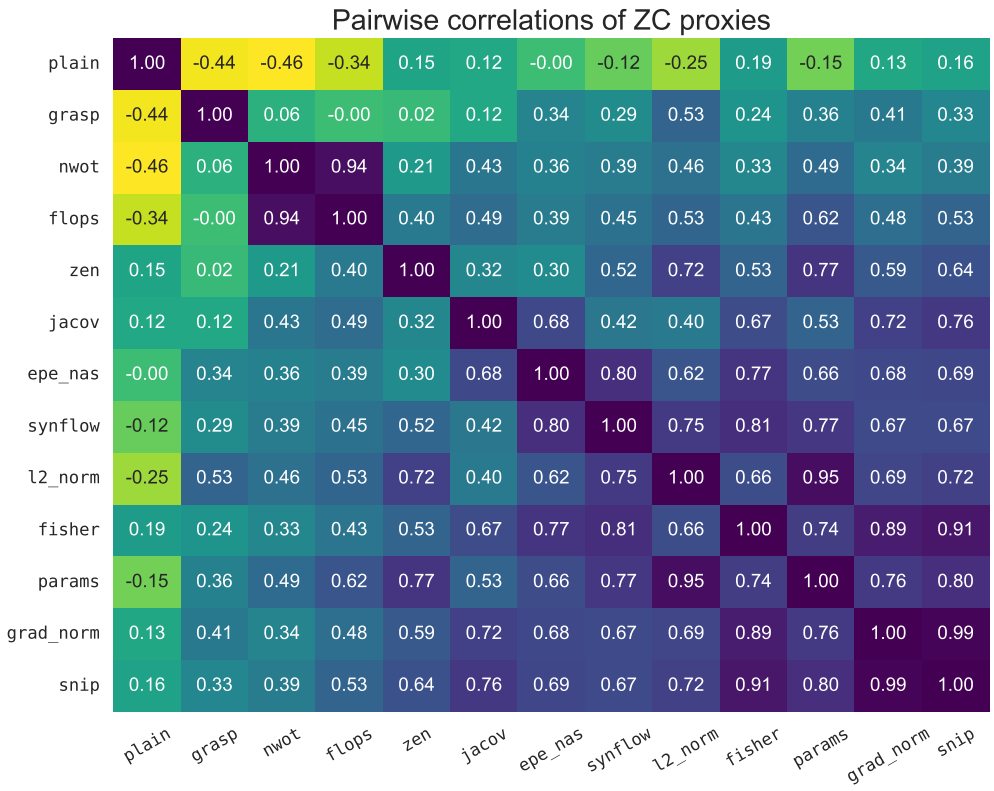

Figure 10: Pearson correlation coefficient for each pair of ZC proxies, averaged over all benchmarks. The entries in the plot are ordered based on the mean score across each row and column.

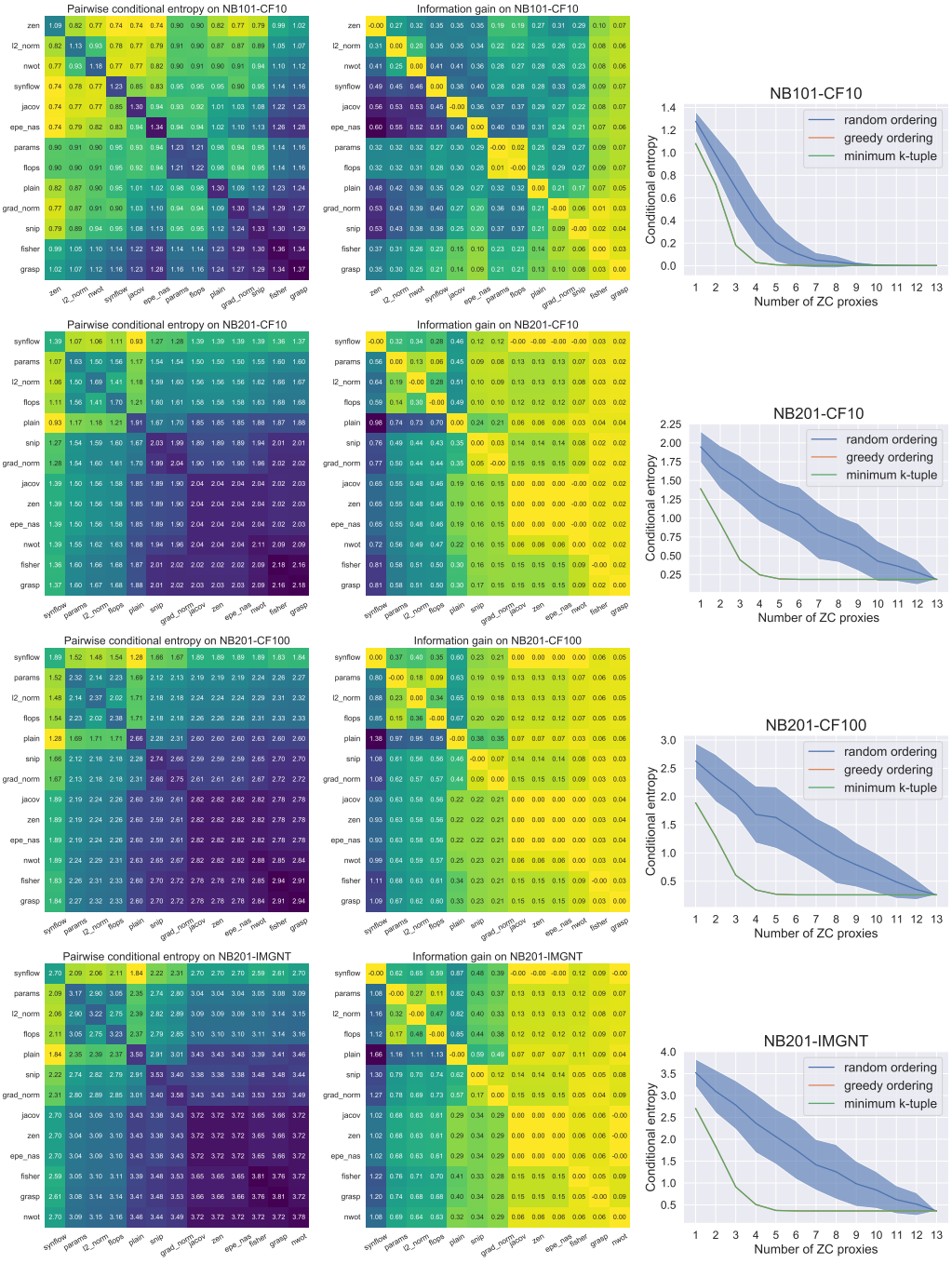

Figure 11: Conditional entropy and information gain (**IG**) for each ZC proxy pair across all search spaces and datasets (Left and Middle). Conditional entropy $H(y \mid z_{i_1}, \ldots, z_{i_k})$ vs. k, where the ordering $z_{i_1}, \ldots, z_{i_k}$ is selected using three different strategies (Right). (1/5)

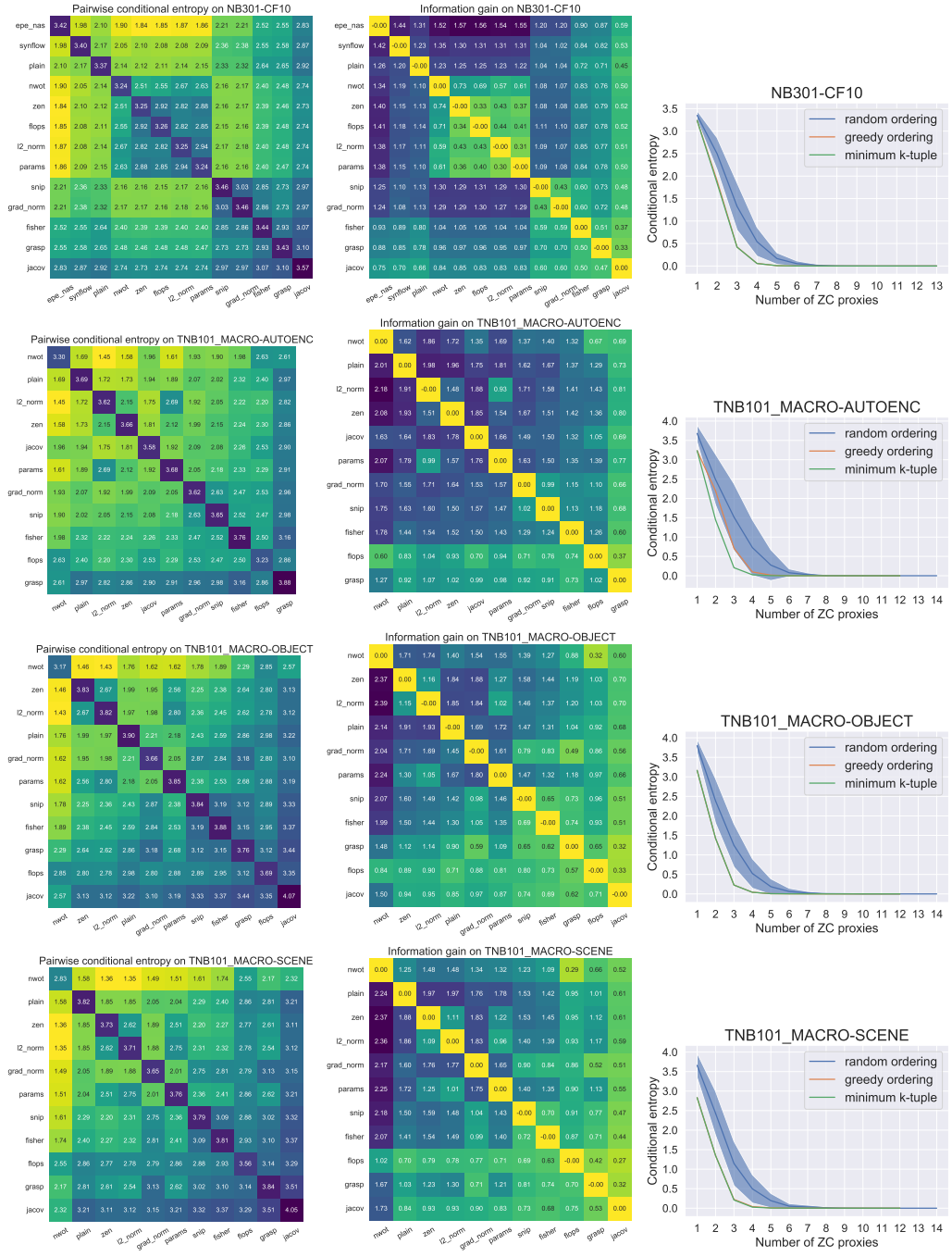

Figure 12: Conditional entropy and information gain (**IG**) for each ZC proxy pair across all search spaces and datasets (Left and Middle). Conditional entropy $H(y \mid z_{i_1}, \ldots, z_{i_k})$ vs. k, where the ordering $z_{i_1}, \ldots, z_{i_k}$ is selected using three different strategies (Right). (2/5)

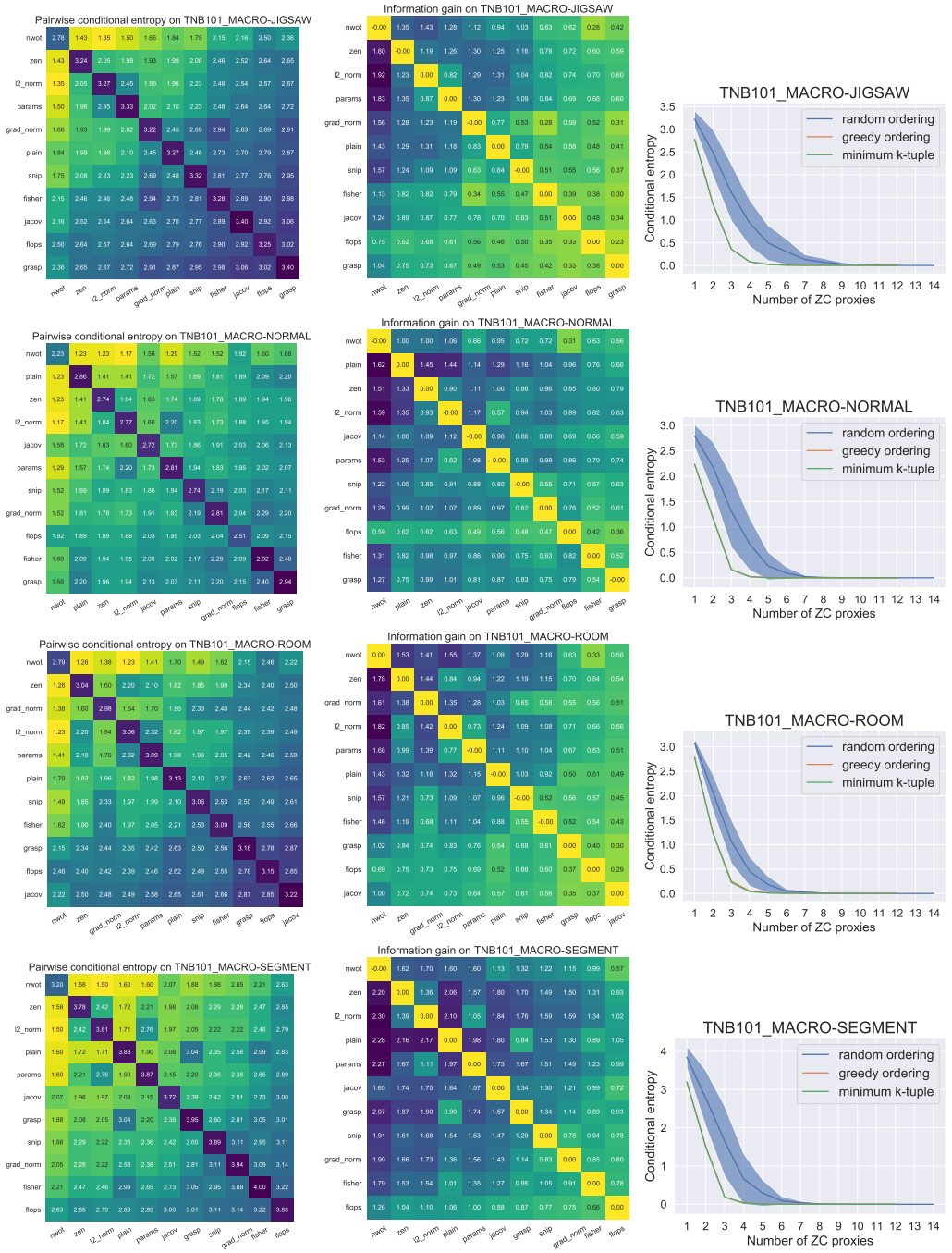

Figure 13: Conditional entropy and information gain (**IG**) for each ZC proxy pair across all search spaces and datasets (Left and Middle). Conditional entropy $H(y \mid z_{i_1}, \ldots, z_{i_k})$ vs. k, where the ordering $z_{i_1}, \ldots, z_{i_k}$ is selected using three different strategies (Right). (3/5)

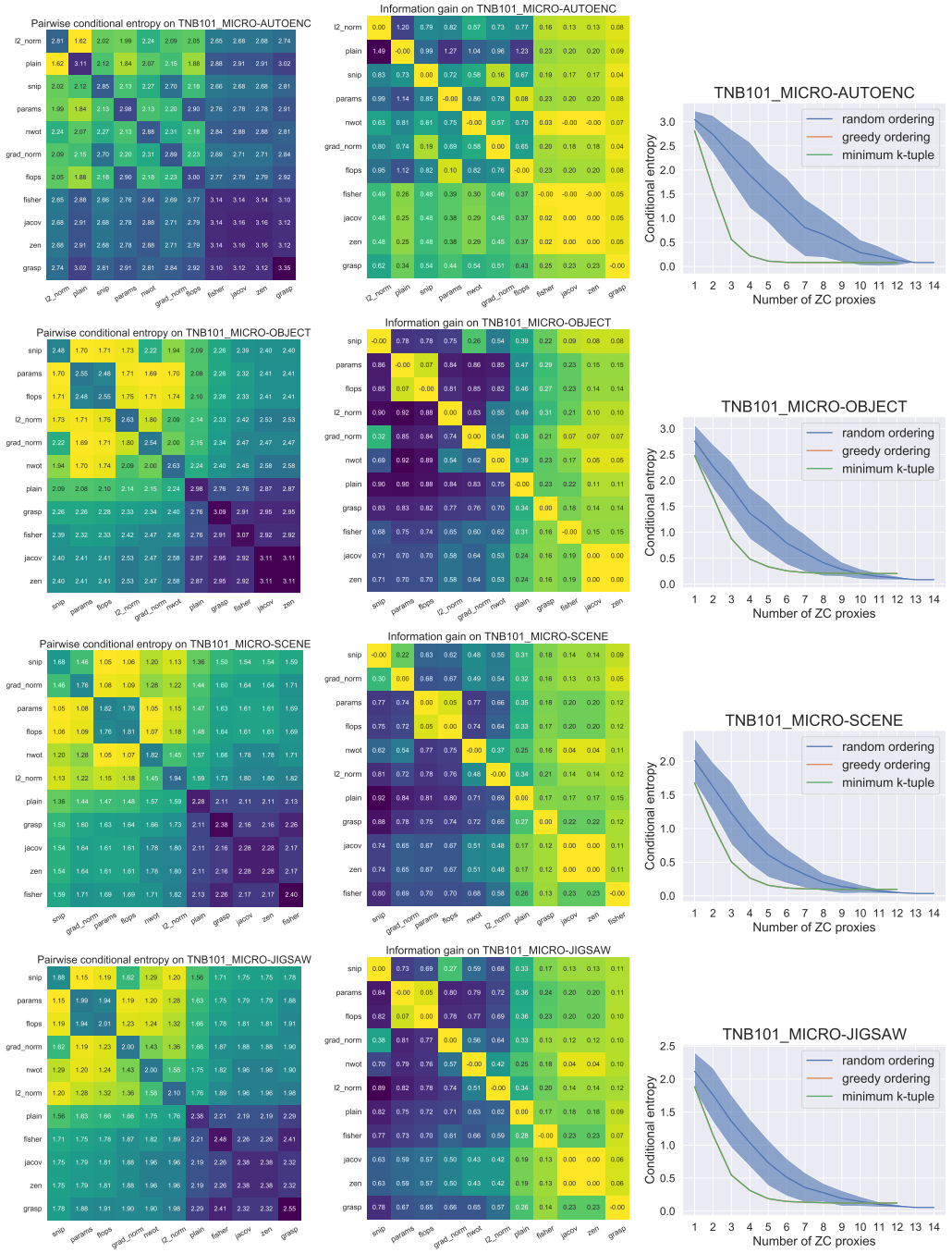

Figure 14: Conditional entropy and information gain (**IG**) for each ZC proxy pair across all search spaces and datasets (Left and Middle). Conditional entropy $H(y \mid z_{i_1}, \ldots, z_{i_k})$ vs. k, where the ordering $z_{i_1}, \ldots, z_{i_k}$ is selected using three different strategies (Right). (4/5)

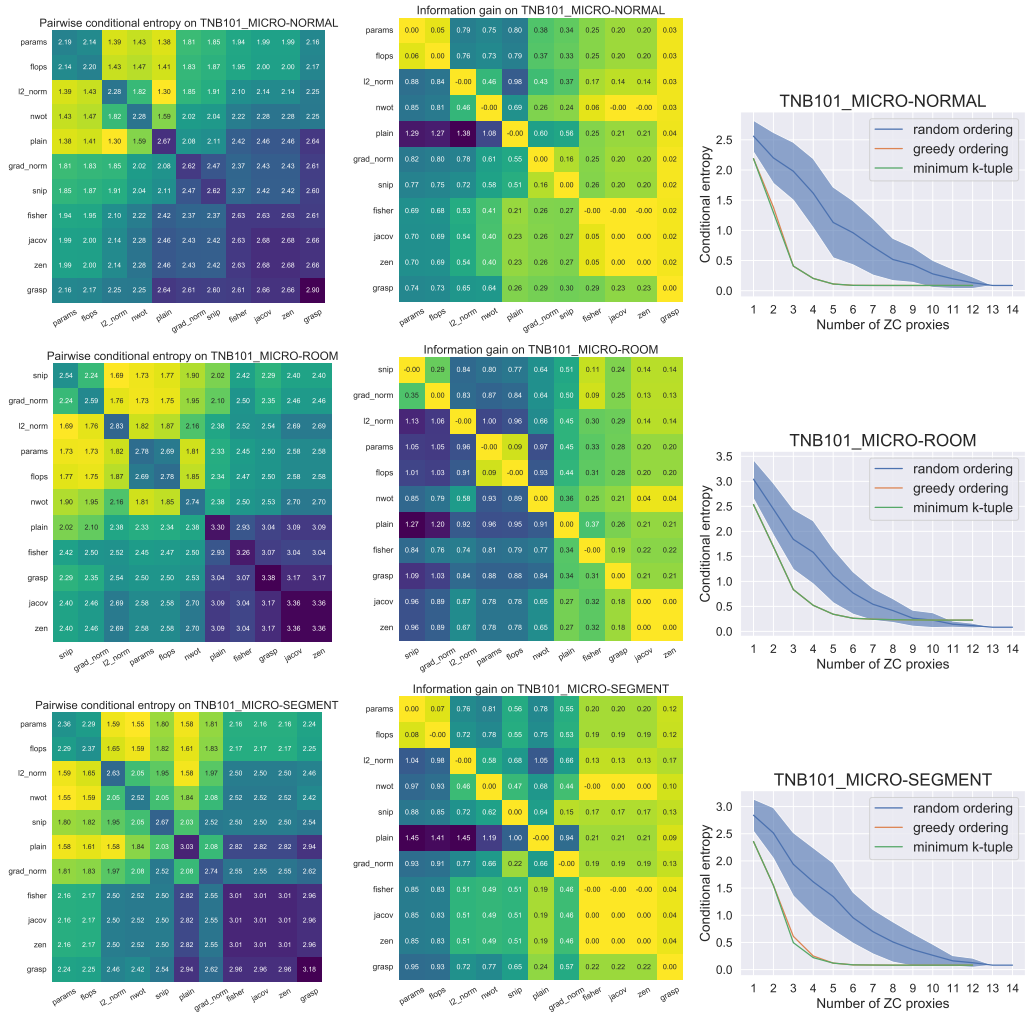

Figure 15: Conditional entropy and information gain (**IG**) for each ZC proxy pair across all search spaces and datasets (Left and Middle). Conditional entropy $H(y \mid z_{i_1}, \ldots, z_{i_k})$ vs. k, where the ordering $z_{i_1}, \ldots, z_{i_k}$ is selected using three different strategies (Right). (5/5)

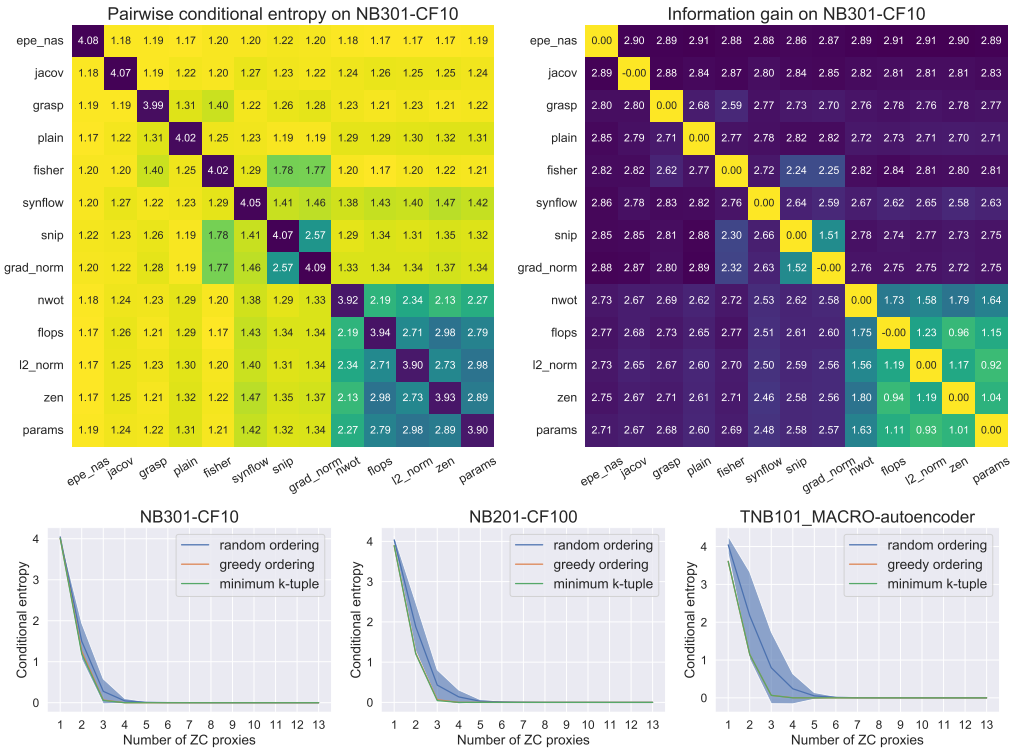

Figure 16: Given a ZC proxy pair (i, j), we compute the conditional entropy $H(y \mid z_i, z_j)$ (top left), and information gain $H(y \mid z_i) - H(y \mid z_i, z_j)$ (top right). Conditional entropy $H(y \mid z_{i_1}, \ldots, z_{i_k})$ vs. k, where the ordering $z_{i_1}, \ldots, z_{i_k}$ is selected using three different strategies. The minimum k-tuple and greedy ordering significantly overlap in the first two figures (bottom). Similar to Figure 4, but using a different bin discretization strategy.

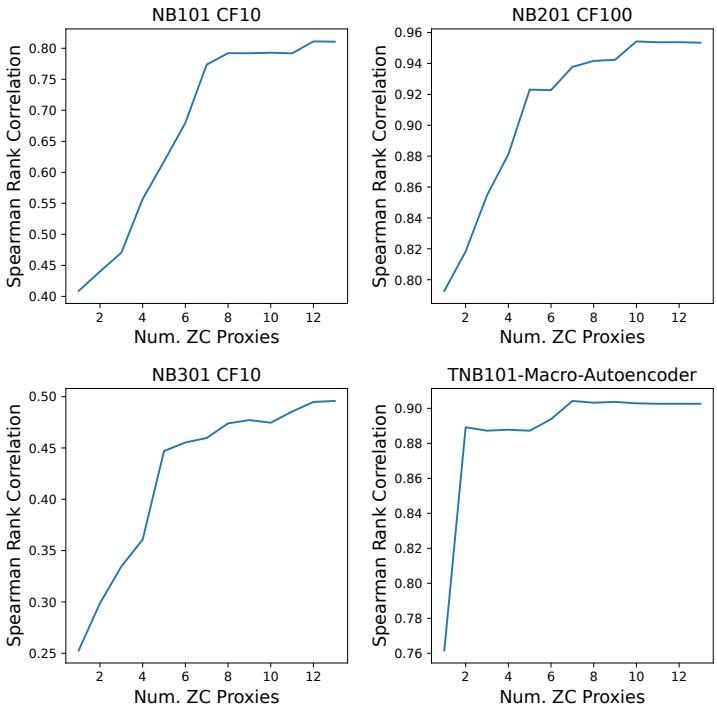

Figure 17: Ablation study on the number of ZC proxies as features vs. rank correlation performance, for an XGBoost surrogate model trained on 1000 randomly drawn architectures. The ordering of ZC proxies is computed via the greedy method from Section 4.3.

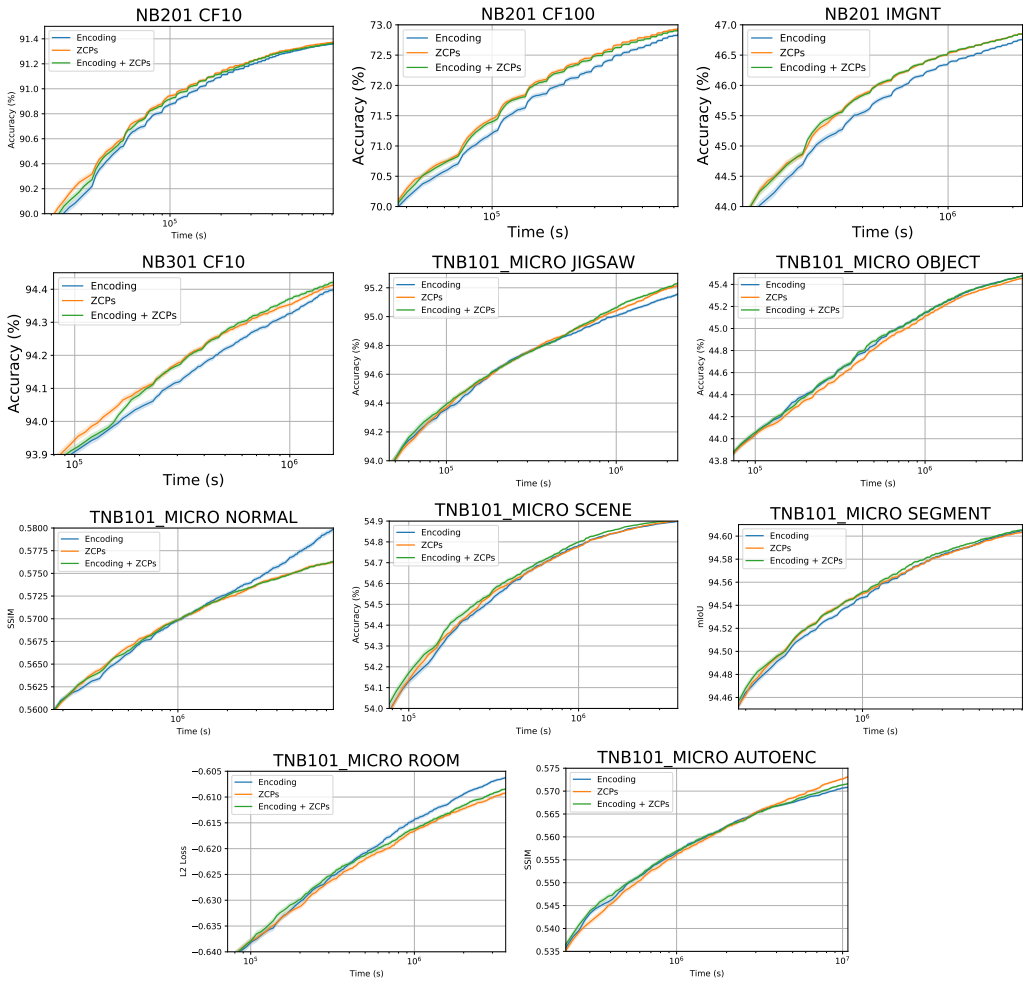

Figure 18: Performance of BANANAS with the vanilla XGBoost surrogate model vs. XGBoost using the additional ZC proxy scores (concatenated to the architecture encoding) as input.

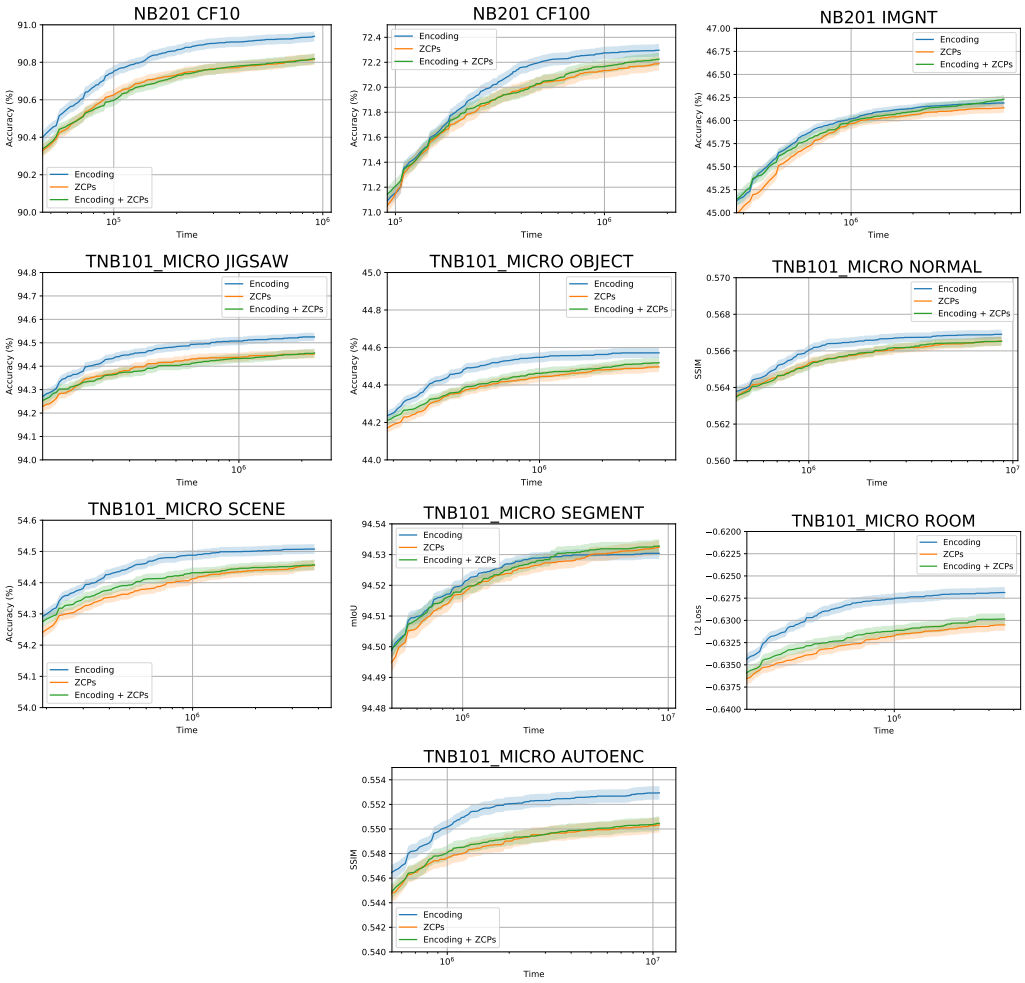

Figure 19: Performance of NPENAS with the vanilla XGBoost surrogate model vs. XGBoost using the additional ZC proxy scores (concatenated to the architecture encoding) as input.

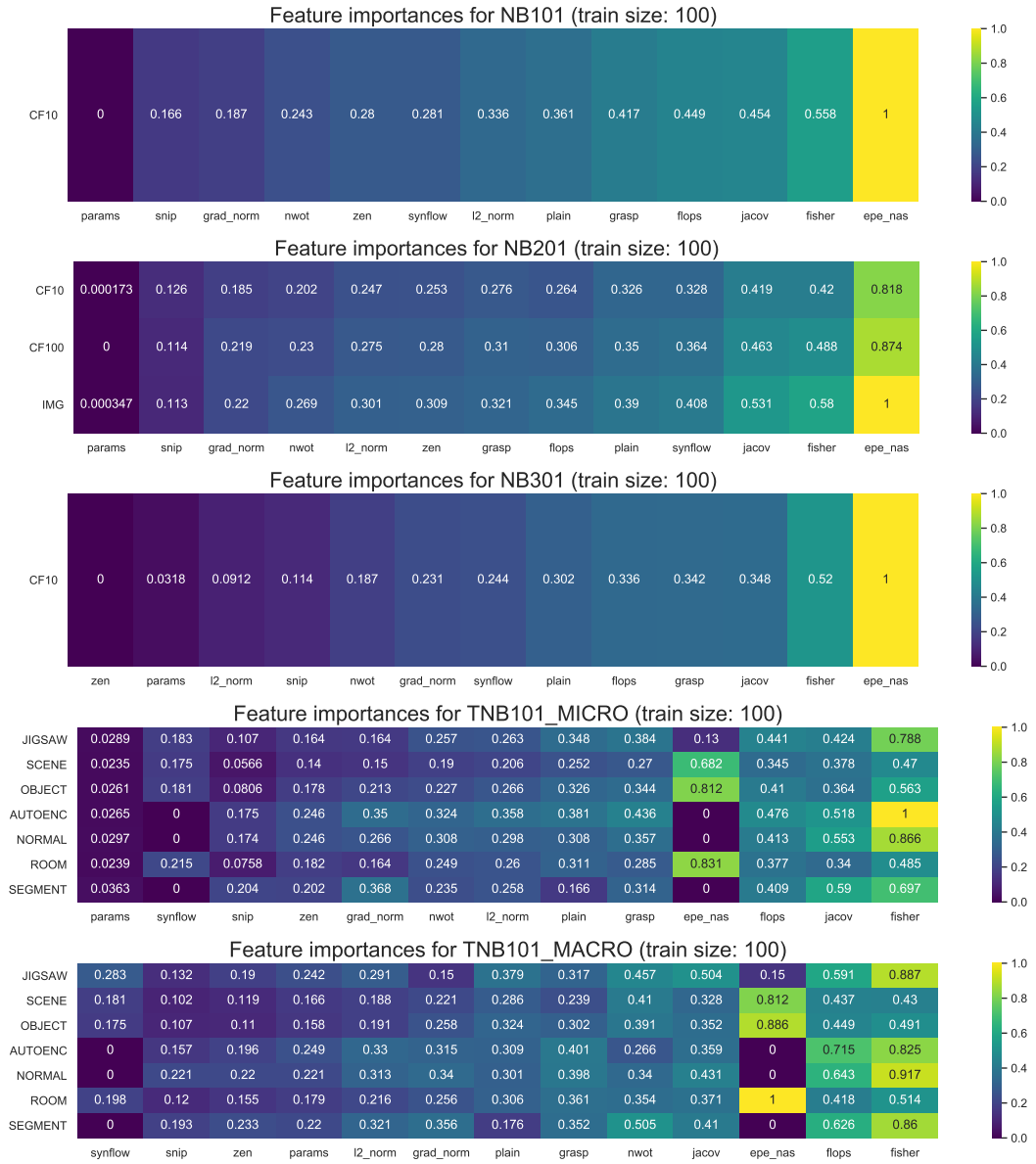

Figure 20: Feature importance values for XGBoost trained on a set of 100 architectures using ZC proxies as features.

Figure 21: Feature importance values for XGBoost trained on a set of 1000 architectures using ZC proxies as features.