What to expect of hardware metric predictors in NAS

Anonymous[1]

[1]Anonymous Institution

Abstract Modern Neural Architecture Search (NAS) focuses on finding the best performing architectures in hardware-aware settings; e.g., those with an optimal tradeoff of accuracy and latency. Due to many advantages of prediction models over live measurements, the search process is often guided by estimates of how well each considered network architecture performs on the desired metrics. Typical prediction models range from operation-wise lookup tables over gradient-boosted trees and neural networks, with little known information on how they compare. We evaluate 18 different performance predictors on ten combinations of metrics, devices, network types, and training tasks, and find that MLP models are the most promising. We then simulate and evaluate how the guidance of such prediction models affects the subsequent architecture selection. Due to inaccurate predictions, the selected architectures are generally suboptimal, which we quantify as an expected reduction in accuracy and hypervolume. We show that simply verifying the predictions of just the selected architectures can lead to substantially improved results. Under a time budget, we find it preferable to use a fast and inaccurate prediction model over accurate but slow live measurements.

1 Introduction

Modern neural network architectures are designed not only considering their primary objective, such as accuracy. While existing architectures can be scaled down to work with the limited available memory and computational power of, e.g., mobile phones, they are significantly outperformed by specifically designed architectures (Howard et al., 2017; Sandler et al., 2018; Zhang et al., 2018; Ma et al., 2018). Standard hardware metrics include memory usage, number of model parameters, Multiply-Accumulate operations, energy consumption, latency, and more; each of which may be limited by the hardware platform or network task. As the range of tasks and target platforms grows, specialized architectures and the methods to find them efficiently are gaining importance.

The automated design and discovery of specialized architectures is the main intent of Neural Architecture Search (NAS). This recent field of study repeatedly broke state of the art records (Zoph et al., 2018; Real et al., 2018; Cai et al., 2019; Tan and Le, 2019; Chu et al., 2019a; Hu et al., 2020) while aiming to reduce the researchers' involvement with this tedious and time-consuming process to a minimum. As the performance of each considered architecture needs to be evaluated, the hardware metrics need to be either measured live or guessed by a trained prediction model. While measuring live has the advantage of not suffering from inaccurate predictions, the corresponding hardware needs to be available during the search process. Measuring on-demand may also significantly slow down the search process and necessitates further measurements for each new architecture search. On the other hand, a prediction model abstracts the hardware from the search code and simplifies changes to the optimization targets, such as metrics or devices. The data set to train the predictor also has to be collected only once so that a trained predictor then works in the absence of the hardware it is predicting for, e.g., in a cloud environment. Furthermore, a differentiable predictor can be used for gradient-based architecture optimization of typically non-differentiable metrics (Cai et al., 2019; Xu et al., 2020; Nayman et al., 2021).

While the many advantages make predictors a popular choice of hardware-aware NAS (e.g. Xu et al. (2020); Wu et al. (2019); Wan et al. (2020); Dai et al. (2020); Nayman et al. (2021)), there

 © 2022 the authors, released under CC BY 4.0

are no guidelines on which predictors perform best, how many training samples are required, or what happens when a predictor is inaccurate. This work investigates the above points. As a first contribution, we conduct large-scale experiments on ten hardware-metric datasets chosen from HW-NAS-Bench (Li et al., 2021a) and TransNAS-Bench-101 (Duan et al., 2021). We explore how powerful the different predictors are when using different amounts of training data and whether these results generalize across different network architecture types. As a second contribution, we extensively simulate the subsequent architecture selection to investigate the impact of inaccurate predictors. Our results demonstrate the effectiveness of network-based prediction models; provide insights into predictor mistakes and what to expect from them. To facilitate reproducibility and further research, our experimental results and code are made available in Appendix A.

2 Related work

NAS Benchmarks. As the search spaces of NAS methods often differ from one another and lack extensive studies, the difficulty of fair comparisons and reproducibility have become a major concern (Yang et al., 2019; Li and Talwalkar, 2020). To alleviate this problem, researchers have exhaustively evaluated search spaces of several thousand architectures to create benchmarks (Ying et al., 2019; Dong and Yang, 2020; Dong et al., 2020; Siems et al., 2020), containing detailed statistics for each architecture. TransNAS-Bench-101 (Duan et al., 2021) evaluates several thousand architectures across seven diverse tasks and finds that the best task-specific architectures may vary significantly.

The popular NAS-BENCH 201 benchmark (Dong and Yang, 2020) has been further extended with ten different hardware metrics for all 15625 architectures. Major findings of this HW-NAS Bench (Li et al., 2021a) include that FLOPs and the number of parameters are a poor approximation for other metrics such as latency, and that hardware-specific costs do not correlate well across hardware platforms. While accounting for each device's characteristics improves the NAS results, it is also expensive. Predictors can reduce costs by requiring fewer measurements and shorter query times. [1].

Predictors in NAS. Aside from real-time measurements (Tan et al., 2019; Yang et al., 2018), hardware metric estimation in NAS is commonly performed via Lookup Table (Wu et al., 2019), Analytical Estimation or a Prediction Model (Dai et al., 2020; Xu et al., 2020). While an operation- and layer-wise Lookup Table can accurately estimate hardware-agnostic metrics, such as FLOPs or the number of parameters (Cai et al., 2019; Guo et al., 2020; Chu et al., 2019a), they may be suboptimal for other metrics with non-obvious and non-linear factors that depend on hardware specifics. Such details can be captured with neural networks (Dai et al., 2020; Mendoza and Wang, 2020; Ponomarev et al., 2020; Xu et al., 2020) or other specialized models (Yao et al., 2018; Wess et al., 2021).

Of particular interest is the correct prediction of the model loss or accuracy, possibly reducing the architecture search time by orders of magnitude (Mellor et al., 2020; Wang et al., 2021; Li et al., 2021b). In addition to common predictors such as Linear Regression, Random Forests (Liaw et al., 2002) or Gaussian Processes (Rasmussen, 2003); specialized techniques may exploit training curve extrapolation, network weight sharing or gradient information. Our experiments follow the recent large-scale study of White et al. (2021), who compare 31 diverse accuracy prediction methods based on initialization and query time, using three NAS benchmarks.

3 Predicting hardware metrics

Our methods follow the large-scale study of White et al. (2021), who compared a total of 31 accuracy prediction methods. The differences between accuracy and hardware-metric prediction, our selection of predictors, and the general training pipeline are described in this section. We then compare these predictors across different training set sizes in our experiments on HW-NAS-Bench and TransNAS-Bench-101, described in Section 4.

[1]For further reading, we recommend a recent survey on hardware-aware NAS (Benmeziane et al., 2021)

Differences to accuracy predictors. There are fundamental differences when predicting hardware metrics and the accuracy of network topologies. The most essential is the cost to obtain a helpful predictor, which may vary widely for accuracy prediction methods. While determining the test accuracy requires the costly and lengthy training of networks, measuring hardware metrics does not necessitate any network training. Consequentially, specialized accuracy-estimation methods that rely on trained networks, loss history, early stopping, gradient statistics or others do not apply to hardware metrics. Therefore, the dominant hardware-metric predictor family is model-based.

Since all relevant predictors are model-based, they can be compared by their training set size. This simplifies the initialization time of a predictor as the number of prior measured architectures on which they are trained. Since an untrained network and a few batches suffice to measure hardware-metrics, the collection of such a training set is comparably inexpensive.

Additionally, hardware predictors are generally used supplementary to a one-shot network optimized for loss or accuracy. Depending on the NAS method, a fully differentiable predictor is required in order to guide the gradient-based architecture selection. Typical choices are Lookup Tables (Cai et al., 2019; Nayman et al., 2021) and neural networks (Xu et al., 2020).

Model-based predictors. The goal of a predictor $f_p(a)$ is to accurately approximate the function $f(a)$, which may be, e.g., the latency of an architecture a from the search space $\mathcal{A}$. A model-based predictor is trained via supervised learning on a set $\mathcal{D}_{train}$ of datapoints $(a, f(a))$, after which it can be inexpensively queried for estimates on further architectures. The collection of the dataset and the duration of the training are referred to as *initialization time* and *training time* respectively.

The quality of such a trained predictor is generally determined by the (ranking) correlation between measurements $\{f(a)|a \in \mathcal{A}_{test}\}$ and predictions $\{f_p(a)|a \in \mathcal{A}_{test}\}$ on the unseen architectures $\mathcal{A}_{test} \subset \mathcal{A}$. Common correlation metric choices are Pearson (PCC), Spearman (SCC) and Kendall's Tau (KT) (Chu et al., 2019b; Yu et al., 2020; Siems et al., 2020).

Our experiments include 18 model-based predictors from different families: Linear Regression, Ridge Regression (Saunders et al., 1998), Bayesian Linear Regression (Bishop, 2007), Support Vector Machines (Cortes and Vapnik, 1995), Gaussian Process (Rasmussen, 2003), Sparse Gaussian Process (Candela and Rasmussen, 2005), Random Forests (Liaw et al., 2002), XGBoost (Chen and Guestrin, 2016), NGBoost (Duan et al., 2020), LGBoost (Ke et al., 2017), BOHAMIANN (Springenberg et al., 2016), BANANAS (White et al., 2019), BONAS (Shi et al., 2020), GCN (Wen et al., 2020), small and large Multi-Layer-Perceptrons (MLP), NAO (Luo et al., 2018), and a layer-operation-wise Lookup Table model. We provide further descriptions and implementation details in Appendix B.

Study setup. We use the NASLib library (Ruchte et al., 2020) for a reliable and large-scale comparison. Each predictor was fit on each dataset and training size 50 times, using seeds $\{0, ..., 49\}$ and normalized input values. See Appendix A for details. We also include hyper-parameter optimization for every predictor, which is detailed in Appendix C.

4 Predictor Experiments

We compare the different predictor models based on two NAS benchmarks, HW-NAS-Bench (Li et al., 2021a) and TransNAS-Bench-101 (Duan et al., 2021). They differ considerably by their network tasks, hardware devices, and architecture designs.

HW-NAS-Bench architecture design and datasets. In HW-NAS-Bench, each architecture is solely defined by the topology of a building block ("cell"), which is stacked multiple times to create a complete network. Each cell is completely defined by choosing six candidate operations. Since they select from five different candidates each time, there are $5^6 = 15625$ unique cell topologies. These cells are not fully sequential but contain paths of different lengths, which is visualized in Appendix D.

HW-NAS-Bench provides ten hardware statistics on CIFAR10, CIFAR100 Krizhevsky et al. (2009) and ImageNet16-120 Chrabaszcz et al. (2017), of which we exclude the incomplete EdgeTPU metric. Thus there are 27 data sets of varying difficulty. As detailed in Appendix E, 12 of them can be accurately fit with Linear Regression and only 25 training samples. Many are also very similar since their measured networks differ only by the number of image classes. We therefore select five datasets that (1.) are not trivial to learn as they are non-linear and (2.) not redundant:

- ImageNet16-120, raspi4, latency
- CIFAR100, pixel3, latency
- CIFAR10, edgegpu, latency
- CIFAR100, edgegpu, energy consumption
- ImageNet16-120, eyeriss, arithmetic intensity

TransNAS-Bench-101 architecture design and datasets. TransNAS-Bench-101 contains information for 7,352 different network architectures, used as backbones in seven diverse vision tasks. Since 4,096 are also a subset of HW-NAS-Bench, we focus on the remaining 3,256 architectures with a macro-level search space. Unlike a micro-level search space, where a cell is stacked multiple times to create a network, each network layer and block is considered individually. In particular, the TransNAS-Bench-101 networks consist of four to six pairs of ResNet blocks (He et al., 2016), which may modify the image size and channels in four ways: not at all, double the channel count, halve the spatial size, and both. Every network has to double the channel count 1 to 3 times, resulting in 3,256 unique architectures. The networks may consequentially differ in their number of layers (depth), the number of channels (width), and image size at any layer.

As done for HW-NAS-Bench and described in Appendix E, we select five of the seven available datasets for their latency measurements.

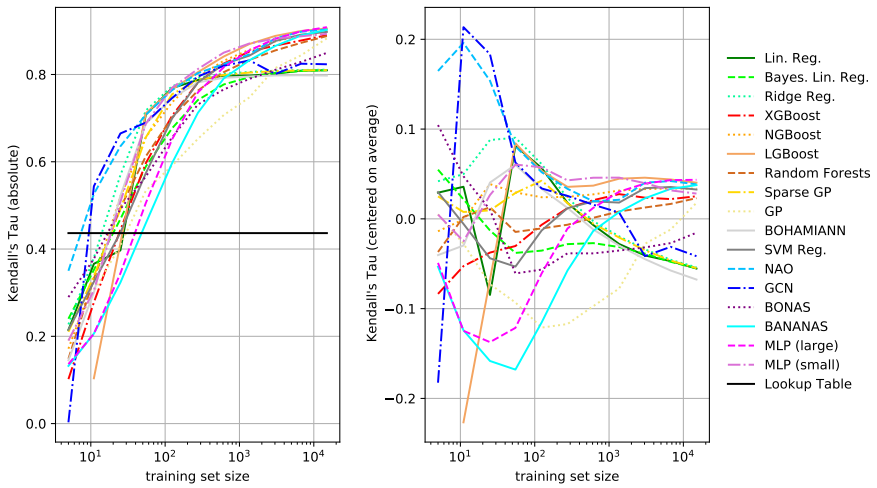
- Object classification
- Scene classification
- Room layout
- Jigsaw
- Semantic segmentation

Fitting results and comparison. The results, averaged over all selected HW-NAS-Bench and TransNAS-Bench-101 datasets, are presented in Figures 1a and 1b, respectively. The left plots present the absolute predictor performance, the right ones make relative comparisons easier.

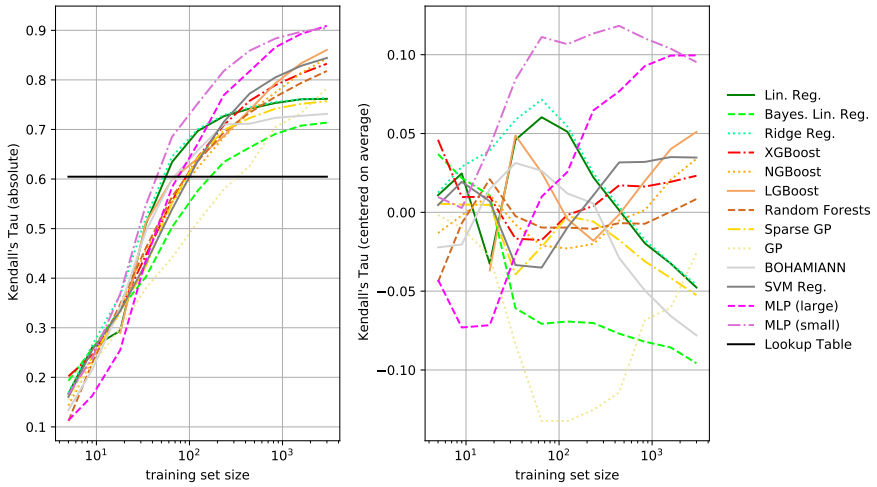
Unsurprisingly, more training samples (i.e., evaluated architectures) generally lead to better prediction results, even until the entire search space is known (aside from the test set). This is true for most of the predictors, although e.g. Gaussian Processes and BOHAMIANN saturate early. The simple Linear Regression and Ridge Regression models also fail to make proper use of hundreds of data points but perform decently when only a few training samples are available. Interestingly, the same is true for the graph-encoding network-based predictors BONAS and GCN. While knowing how the different paths within each cell connect (see Appendix B) is especially useful given less than fifty training samples, the advantage disappears afterward. In contrast, the graph-encoding encoder-decoder approach of NAO performs decently at all times.

Due to their powerful rule-based approach, tree-based models perform much better given many training samples. Under such circumstances, LGBoost is a candidate for the best predictor model. Similarly, the predictions of Support Vector Machines also benefit strongly from more samples.

The model we find to perform best for most training set sizes are MLPs. They are among the top predictors at almost all times in the HW-NAS-Bench, although tree-based models are competitive given enough data. After around 3,000 training samples, thinner and deeper MLPs improve over the wider and smaller ones. The path-encoding BANANAS model behaves similarly to a regular large MLP but requires more samples to reach the same performance. This is interesting since, aside from the data encoding, BANANAS is an ensemble of three large MLP models. Even though only the first network layer is affected by the data encoding, the more complicated path-encoding proves harmful when the connectivity of the architectures in the search space is fixed. On TransNAS-Bench-101, MLP perform exceptionally well. They are much better than any other tested predictor once more than just 20 training samples are available. The small MLP model can achieve a KT correlation of



(a) Results on HW-NAS-Bench. NAO performs decently at all times, and none of the prediction models requires more than 60 training samples to improve over a Lookup Table model.



(b) Results on TransNAS-Bench-101. Since all network architectures are purely sequential by design, we do not evaluate predictors that specifically encode the architecture connectivity (BANANAS, BONAS, GCN, NAO). After as few as 20 training samples, MLP models outclass all other predictors.

Figure 1: How well the different predictors rank the test architectures, depending on the training set size and averaged over the five selected datasets. **Left plots**: absolute Kendall's Tau ranking correlation, higher is better. **Right plots**: same as left, but centered on the predictor-average.

80% with just 200 training samples, which takes the best non-network-based predictor (Support Vector Machine) four times as many. They are also the only models that achieve a KT correlation of over 90%, about 5% higher than the next best model (LGBoost).

Finally, the Lookup Table models (black horizontal lines) perform poorly in comparison to any other predictor. Even though building such a model for HW-NAS-Bench datasets requires only 25 neighbored architectures, NAO and GCN perform better after just ten random samples. More than half of the predictor models require less than 25 random samples, while the worst need at most 60. On TransNAS-Bench-101, Lookup Tables perform comparably better. Building one requires only 21 neighbored architectures, and it takes most models between 50 and 100 random training samples to achieve better performance. When measured on a per dataset basis, we find that the Lookup Table models display a severe performance difference ranging from about 20% KT correlation (cifar10-edgegpu_latency and Jigsaw) to over 70% (ImageNet16-120-eyeriss_arithmetic_intensity and Semantic Segmentation, see Appendix E). Other models prove to be much more stable.

Table 1: The Kendall's Tau correlation of Lookup Tables and Linear Regression (in brackets, using only 124 training samples) across metrics and devices. More details are available in Appendix E.

	HW-NAS-Bench					TransNAS-101
	Raspi4	FPGA	Eyeriss	Pixel3	EdgeGPU	Tesla V100
latency	0.45 (0.75)	0.99 (0.97)	0.99 (0.96)	0.49 (0.78)	0.21 (0.79)	0.60 (0.70)
energy		0.99 (0.97)	1.00 (0.99)		0.23 (0.79)	
arithmetic_intensity			0.84 (0.81)			

Devices and Metrics. The previously described results are based on a specific selection of HW-NAS-Bench and TransNAS-Bench-101 datasets that are hard to fit for Lookup Table models. As shown in Table 1, that is not always the case. The FPGA and Eyeriss hardware devices are very suitable for Lookup Tables, so that achieving an almost perfect ranking correlation is possible. Nonetheless, Linear Regression requires only 124 training samples to compete even there and is significantly better in every other case. We finally observe that the difficulty of fitting predictors primarily depends on the hardware device, much more than the measured metric.

5 Evaluating the predictor-guided architecture selection

Although the experiments in Section 4 greatly assist us in selecting a predictor, it is not clear what a specific Kendall's Tau correlation implies for the subsequent architecture selection. Given a perfect hardware metric predictor (Kendall's Tau = 1.0), we can expect that an ideal architecture search process will select the architectures with the best tradeoff of accuracy and the hardware metric, i.e., the true Pareto front. On the other hand, imperfect predictions result in the selection of supposedly-best architectures that are wrongly believed to be better.

To study how hardware predictors affect NAS results, we extensively evaluate the selection of such supposedly-best architectures in simulation. This approach can evaluate any combination of predictor quality, test set size, and dataset, without the technical difficulties of obtaining actual predictor models that precisely match such requirements. Since the hardware and accuracy prediction models are usually independent and can be studied in isolation, we use ground-truth accuracy values in all cases.

Simulating predictors. The main challenge of the simulation is to quickly and accurately model predictor outputs. We base our simulation on how predictor-generated values deviate from their ground-truth targets on the test set, which is explained in Figure 2 and further detailed in Appendix G. Since the simulated deviations are similar to those of actual predictors, simulated predictions are obtained by drawing random values from this deviation distribution and adding them to the ground-truth hardware measurements.

A single example of a simulation can be seen in Figure 3. Although most selected architectures (orange) are close to the true optimum (red Pareto front), there almost always exists an architecture that has superior accuracy and, at most, the same latency. Simply accepting the 13 selected architectures in this particular example results in a mean reduction of accuracy (MRA_{all}) of 1.06%. In other words, the average selected architecture has 1.06% lower accuracy than a comparable one on the true Paret front. However, simply verifying the hardware metric predictions through actual measurements reveals that some selected architectures are suboptimal. By choosing only the Pareto subset of the selection, the opportunity loss can be reduced to 0.43% (MRA_{pareto}).

An important property of this approach is that it is independent of any particular optimization method. The supposedly-best architectures are always correctly identified, which is an upper bound on how well Bayesian Optimization, Evolutionary Algorithms, and other approaches can perform. The exemplary MRA_{pareto} opportunity loss of 0.43% is therefore *unavoidable* and depends solely on the hardware metric predictor, the dataset, and the number of considered architectures.

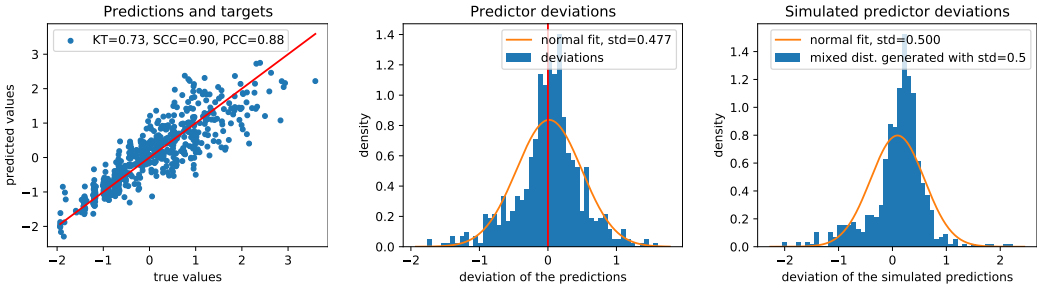


Figure 2: A trained XGBoost prediction model on normalized ImageNet16-120 raspi4-latency test data. **Left**: The latency prediction (y-axis) for every architecture (blue dot) is approximately correct (red line). **Center**: The same data as on the left, the distribution of deviations made by the predictor (blue) and a normal distribution fit to them (orange). **Right**: A mixed distribution can simulate typical deviation distributions as that in the center plot.

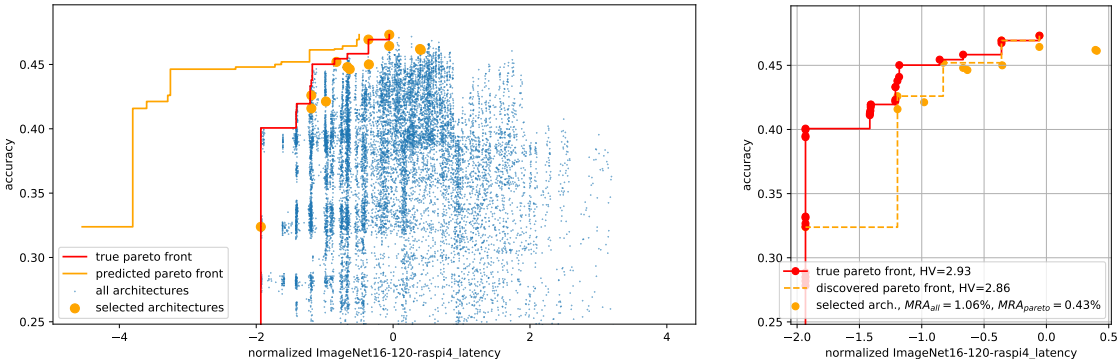


Figure 3: An example of predictor-guided architecture selection, std=0.5. **Left**: The simulated predictor makes an inaccurate latency prediction for each architecture (blue), resulting in the selection of the supposedly-best architectures (orange dots). Even though the predicted Pareto front (orange line) may differ significantly from the ground-truth Pareto front (red line), most selected architectures are close to optimal. **Right**: Same data as left. Simply accepting all selected architectures results in a Mean Reduction of Accuracy (*MRA*) of 1.06%, while verifying the predictions and discarding inferior results improves that to 0.43%. The hypervolume (HV, area under the Pareto-fronts) is reduced by 0.07.

Results. We simulate 1,000 architecture selections for each of the five chosen HW-NAS-Bench datasets, six different test set sizes, and eleven distribution standard deviations between 0.0 and 1.0. As exemplarily shown in Figure 3, each such simulation allows us to compute the mean reduction in accuracy (MRA) and the hypervolume (HV) under the Pareto fronts. The most important insights are visualized in Figure 4 and summarized below.

Verifying the predicted results matters (Figure 4, left). The best prediction models achieve a KT correlation of almost 0.9, which translates to a mean reduction in accuracy of MRA_{all} ≈1.5%. That means, for each selected architecture, there exists an architecture of equal or lower latency in the true Pareto set (if latency is the hardware metric) that improves the average accuracy by 1.5%. Even though all selected architectures are believed to form a Pareto set, that is not the case. Their optimal subset has a reduction of only $\text{MRA}_{pareto} \approx 0.5\%$, a significant improvement. However, finding this optimal subset requires actually measuring the hardware metrics of the architectures selected by the used NAS method.

Furthermore, the left of Figure 4 aids in anticipating the MRA given a specific predictor. If one used e.g. BOHAMIANN (KT≈0.8, see Figure1a) instead of MLPs or LGBoost (KT≈0.9), MRA_{pareto} increases from around 0.5% to roughly 1.2%. The average accuracy of the selected architectures is thus reduced by another 0.7%, just by using an unsuitable hardware metric predictor. Lookup Tables (KT≈0.45) are not even visualized anymore, they have an MRA_{pareto} of over 2.5%.

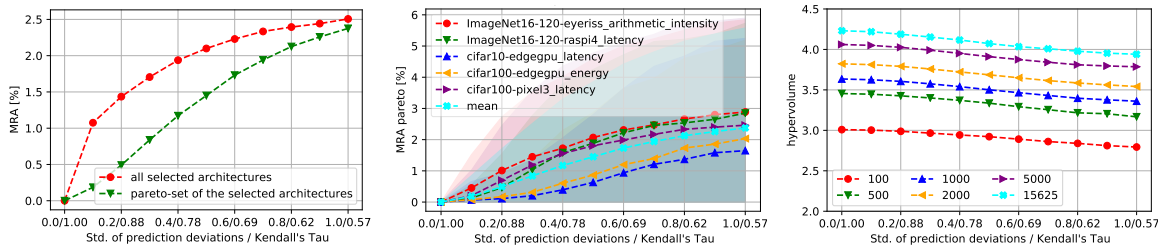


Figure 4: Simulation results, with the standard deviation of the predictor deviations and the resulting KT correlation on the x-axis. **Left**: Verifying the hardware predictions can significantly improve the results, even more so for better predictors. **Center**: The drops in average accuracy are dependant on the dataset and hardware metric. **Right**: Considering more candidate architectures and using better prediction models results in a larger hypervolume.

Another interesting observation is that the gap between MRA_{all} and MRA_{pareto} is wider for better predictors. This is a shortcoming of the MRA metric that we elaborate on in Appendix H.

The dataset and metric matter (Figure 4, center). While we generally present the results averaged over datasets, there exists some discrepancy among them. Most interestingly, predicting hardware metrics on harder classification problems (ImageNet16-120 is harder than CIFAR10) also results in a higher MRA. This is especially important since MRA is an absolute accuracy reduction. Even though the CIFAR10 networks achieve twice the accuracy of ImageNet16-120 networks, they lose less absolute accuracy to imperfect predictions. The order of MRA/dataset is primarily stable for any predictor KT correlation. Finally, as visualized by the shaded areas, the standard deviation of the MRA is huge. Consequentially, predictor-guided NAS is very likely to produce results of varying quality for each different predictor or search attempt, especially with less accurate predictors.

The number of considered architectures matters (Figure 4, right). We measure the hypervolume of the discovered Pareto front (i.e., the area beneath it, see Appendix H), which, unlike MRA, also considers the hardware metric. Quite obviously, if the architectures from the true Pareto set are not considered, they can not be selected. To achieve the highest possible hypervolume of around 4.2 (i.e. find the true Pareto set), every architecture in the search space must be evaluated with a perfect predictor. This is impossible in most real-world cases, where only a tiny fraction of all possible architectures can ever be considered.

For HW-NAS-Bench, considering 5000 architectures with perfect live measurements and predicting the metrics for all 15625 with ranking correlation KT≈0.73 results in selecting equivalent sets of architectures. As seen in Figure 1a, Ridge Regression can achieve this performance with fewer than 100 training samples. Thus, a worse predictor leads to better results if it enables considering more architectures. This insight is especially crucial for live measurements, which are accurate but slow. Similarly, estimating the network accuracy with super-networks takes much more time than predicting their performance with a neural predictor (Wen et al., 2020). If the measurement of any metrics is the limiting factor, a guided selection of a cheap predictor is likely to do better.

6 Discussion

Chosen prediction methods. Given the nature of hardware-metric prediction, only the subset of model-based predictors evaluated by White et al. (2021) is suitable. We extended this subset with four models, including the popular Lookup Table. We abstained from evaluating layer-wise predictors (e.g. Wess et al. (2021)) since such data is not available, and meta-learning predictors (Lee et al., 2021) due to the vast possibilities to configure them. A separate and specialized comparison between classic and meta-learning predictors seems favorable to us.

Simulation limitations. In contrast to evaluating real predictors, the simulation allows us to quickly make statements for any test set sizes and predictor-inaccuracies. However, naturally, the results

are only approximations. While they match actual values, they are generally slightly pessimistic (see Appendix I). We also limit the simulation to HW-NAS-Bench since the changes to classification results are more accessible to interpretation than changes to loss values across different problem types. Finally, the current simulation approach does not consider specific architecture selection methods but an upper bound of always selecting optimally. This choice highlights that both the average accuracy and the hypervolume will be reduced for inaccurate predictors independently of the used optimization method. Nonetheless, a detailed study of how individual methods approach the upper bound is an interesting direction for future research.

Broader Impact. We hope that our study helps understanding and improving AutoML in general. Still, even with improved device-aware results, we do not believe AutoML to make ML engineering obsolete in the foreseeable future. For better or worse, AutoML is however capable of accelerating and magnifying the impact of many other machine learning fields.

Transferability of the results. Our evaluation includes five challenging and diverse datasets based on the micro-level search space of HW-NAS-Bench and five latency-based datasets of various macro-level search space architectures in TransNAS-Bench-101. Nonetheless, we find shared trends: All tested prediction models improve over Lookup Tables with little amounts of training data. Furthermore, most predictors benefit from more training data, even until the entire search space (aside from the test set) is known. We also find that network-based predictors are generally best but may be challenged by tree-based predictors if enough training data is available. Given only a few samples, Ridge Regression performs better than most other models.

Recommendations. While Lookup Tables are a cheap, simple, and popular model in gradient-based architecture selection, we find a significant variance in performance across tasks and devices (see Table 1 and Appendix E). We recommend replacing such models with either MLPs or Ridge Regression, which are more stable, fully differentiable, and often take less than 100 training samples to achieve better results.

For most realistic scenarios where more than 100 training samples are available, MLP models are the most promising. They are among the top predictors on HW-NAS-Bench and demonstrate outstanding performance on the TransNAS-Bench-101 datasets. We found that specialized architecture encodings are primarily beneficial for little training data but suspect that they enjoy an additional advantage when network topologies are more complex and diverse (White et al., 2021).

While the query time for all predictors is less than 0.05s and thus negligible, there is a notable difference in training time (see Appendix F). We recommend Ridge Regression for tiny amounts of training data and LGBoost otherwise if that is an essential factor.

If a NAS method selects architectures based on hardware metric predictions, we strongly suggest verifying the results by measuring the true metric value afterward. Doing so may eliminate inferior candidates and improve the average result substantially. Finally, if the limiting factor to a particular NAS method is the slow live measurement of hardware metrics, using a much faster predictor may lead to an improvement, even if the prediction model is less accurate.

7 Conclusions

This work evaluated various hardware-metric prediction models on ten problems of different metrics, devices, and network architecture types. We then simulated the selection process for different test set sizes and predictor inaccuracies to improve our understanding of predictor-based architecture selection. We show quantitatively that even imperfect predictors may improve NAS if their low query time enables considering more candidate architectures. Finally, verifying the predictions for the few selected candidates can prevent using inferior architectures that are wrongly believed to be better, notably improving the average performance. The code and results are made available, thus acting both for recommendation and as a baseline for future works.

8 Reproducibility 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]

   (b) Did you describe the limitations of your work? [Yes] Please see Section 6.

   (c) Did you discuss any potential negative societal impacts of your work? [No] As this paper is focused on basic NAS research and understandung, we do not believe such a discussion beneficial.

   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results... [N/A]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] We have added our code and results, and the instuctions how to run each part of the experiments (including plotting) in the *README.md*. Our Appendices detail design and hyper-parameter choices.

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] While we do not provide the thousands of json files creating by running that many experiments using NASLib, we provide an automatically generated aggregation of the results as csv file, as well as the code for this.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes]

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] These details are available in our appendices, as well as the NASLib documentation.

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] We ran our experiments with NASLib and inspected the run configurations at random, to see whether they were produced as intended.

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We believe that an ablation study only applies to the simulation of predictor mistakes and have prepared a sanity check in Appendix I.

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes]

   (i) Did you compare performance over time? [N/A]

   (j) Did you perform multiple runs of your experiments and report random seeds? [Yes] Yes and No. We run predictors in each setting 50 times, however on the seeds $\{0..49\}$. Our simulation performs each experiment 1000 times with fully random seeds.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We lack the required space to do so for most plots. However, since all results and the plotting code are made available, we do not think this to be a problem.

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] Our results are based on HW-NAS-Bench and TransNAS-Bench-101.

(m) 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 mentioned this in Appendix A.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] This is answered in Appendix C.

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] Our work is mostly based on HW-NAS-Bench, TransNAS-Bench-101, and NASLib, all of which are cited.

(b) Did you mention the license of the assets? [Yes] We give an overview in Appendix A.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide code supplementing NASLib, lookup files for HW-NAS-Bench and TransNAS-Bench-101 results, and our own results.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects… [N/A]

References

Benmeziane, H., Maghraoui, K. E., Ouarnoughi, H., Niar, S., Wistuba, M., and Wang, N. (2021). A Comprehensive Survey on Hardware-Aware Neural Architecture Search. *CoRR*, abs/2101.09336.

Bishop, C. M. (2007). *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer.

Cai, H., Zhu, L., and Han, S. (2019). ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Candela, J. Q. and Rasmussen, C. E. (2005). A Unifying View of Sparse Approximate Gaussian Process Regression. *J. Mach. Learn. Res.*, 6:1939–1959.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*.

Chu, X., Zhang, B., Li, J., Li, Q., and Xu, R. (2019a). ScarletNAS: Bridging the Gap Between Scalability and Fairness in Neural Architecture Search. *CoRR*, abs/1908.06022.

Chu, X., Zhang, B., Xu, R., and Li, J. (2019b). FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. *CoRR*, abs/1907.01845.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Dai, X., Wan, A., Zhang, P., Wu, B., He, Z., Wei, Z., Chen, K., Tian, Y., Yu, M., Vajda, P., and Gonzalez, J. E. (2020). FBNetV3: Joint Architecture-Recipe Search using Neural Acquisition Function. *CoRR*, abs/2006.02049.

Dong, X., Liu, L., Musial, K., and Gabrys, B. (2020). NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size. *arXiv preprint arXiv:2009.00437*.

Dong, X. and Yang, Y. (2020). NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Duan, T., Anand, A., Ding, D. Y., Thai, K. K., Basu, S., Ng, A. Y., and Schuler, A. (2020). Ngboost: Natural gradient boosting for probabilistic prediction. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 2690–2700. PMLR.

Duan, Y., Chen, X., Xu, H., Chen, Z., Liang, X., Zhang, T., and Li, Z. (2021). TransNAS-Bench-101: Improving Transferability and Generalizability of Cross-Task Neural Architecture Search. *CoRR*, abs/2105.11871.

Gencoglu, O., van Gils, M. J., Guldogan, E., Morikawa, C., Süzen, M., Gruber, M., Leinonen, J., and Huttunen, H. (2019). HARK Side of Deep Learning - From Grad Student Descent to Automated Machine Learning. *CoRR*, abs/1904.07633.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. (2020). Single Path One-Shot Neural Architecture Search with Uniform Sampling. In *European Conference on Computer Vision*, pages 544–560. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision applications. *CoRR*, abs/1704.04861.

Hu, S., Xie, S., Zheng, H., Liu, C., Shi, J., Liu, X., and Lin, D. (2020). DSNAS: Direct Neural Architecture Search without Parameter Retraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12084–12092.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3146–3154.

Krizhevsky, A., Nair, V., and Hinton, G. (2009). CIFAR-10 (Canadian Institute for Advanced Research).

Lee, H., Lee, S., Chong, S., and Hwang, S. J. (2021). HELP: Hardware-Adaptive Efficient Latency Predictor for NAS via Meta-Learning. *CoRR*, abs/2106.08630.

Li, C., Yu, Z., Fu, Y., Zhang, Y., Zhao, Y., You, H., Yu, Q., Wang, Y., and Lin, Y. (2021a). HW-NAS-Bench: Hardware-Aware Neural Architecture Search Benchmark. *CoRR*, abs/2103.10584.

Li, G., Mandal, S. K., Ogras, Ü. Y., and Marculescu, R. (2021b). FLASH: Fast Neural Architecture Search with Hardware Optimization. *CoRR*, abs/2108.00568.

Li, L. and Talwalkar, A. (2020). Random Search and Reproducibility for Neural Architecture Search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR.

Liaw, A., Wiener, M., et al. (2002). Classification and Regression by randomForest. *R news*, 2(3):18–22.

Lindauer, M. and Hutter, F. (2020). Best Practices for Scientific Research on Neural Architecture Search. *Journal of Machine Learning Research*, 21(243):1–18.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T. (2018). Neural Architecture Optimization. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7827–7838.

Ma, N., Zhang, X., Zheng, H., and Sun, J. (2018). ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIV*, volume 11218 of *Lecture Notes in Computer Science*, pages 122–138. Springer.

Mellor, J., Turner, J., Storkey, A., and Crowley, E. J. (2020). Neural Architecture Search without Training.

Mendoza, D. M. and Wang, S. (2020). Predicting Latency of Neural Network Inference.

Nayman, N., Aflalo, Y., Noy, A., and Zelnik, L. (2021). HardCoRe-NAS: Hard Constrained diffeRentiable Neural Architecture Search. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 7979–7990. PMLR.

Ponomarev, E., Matveev, S. A., and Oseledets, I. V. (2020). LETI: Latency Estimation Tool and Investigation of Neural Networks inference on Mobile GPU. *CoRR*, abs/2010.02871.

Rasmussen, C. E. (2003). Gaussian Processes in Machine Learning. In Bousquet, O., von Luxburg, U., and Rätsch, G., editors, *Advanced Lectures on Machine Learning, ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, volume 3176 of *Lecture Notes in Computer Science*, pages 63–71. Springer.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2018). Regularized Evolution for Image Classifier Architecture Search.

Ruchte, M., Zela, A., Siems, J., Grabocka, J., and Hutter, F. (2020). Naslib: A modular and flexible neural architecture search library. `https://github.com/automl/NASLib`.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

Saunders, C., Gammerman, A., and Vovk, V. (1998). Ridge Regression Learning Algorithm in Dual Variables. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, page 515–521, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Shi, H., Pi, R., Xu, H., Li, Z., Kwok, J. T., and Zhang, T. (2020). Bridging the Gap between Sample-based and One-shot Neural Architecture Search with BONAS. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Siems, J., Zimmer, L., Zela, A., Lukasik, J., Keuper, M., and Hutter, F. (2020). NAS-Bench-301 and the Case for Surrogate Benchmarks for Neural Architecture Search.

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. (2016). Bayesian Optimization with Robust Bayesian Neural Networks. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4134–4142.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 2820–2828. Computer Vision Foundation / IEEE.

Tan, M. and Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *CoRR*, abs/1905.11946.

Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K., Vajda, P., and Gonzalez, J. E. (2020). FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 12962–12971. IEEE.

Wang, R., Chen, X., Cheng, M., Tang, X., and Hsieh, C. (2021). RANK-NOSH: Efficient Predictor-Based Architecture Search via Non-Uniform Successive Halving. *CoRR*, abs/2108.08019.

Wen, W., Liu, H., Chen, Y., Li, H. H., Bender, G., and Kindermans, P. (2020). Neural Predictor for Neural Architecture Search. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J., editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXIX*, volume 12374 of *Lecture Notes in Computer Science*, pages 660–676. Springer.

Wess, M., Ivanov, M., Unger, C., Nookala, A., Wendt, A., and Jantsch, A. (2021). ANNETTE: Accurate Neural Network Execution Time Estimation With Stacked Models. *IEEE Access*, 9:3545–3556.

White, C., Neiswanger, W., and Savani, Y. (2019). BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. *arXiv preprint arXiv:1910.11858*.

White, C., Zela, A., Ru, B., Liu, Y., and Hutter, F. (2021). How Powerful are Performance Predictors in Neural Architecture Search? *CoRR*, abs/2104.01177.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. (2019). FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 10734–10742. Computer Vision Foundation / IEEE.

Xu, Y., Xie, L., Zhang, X., Chen, X., Shi, B., Tian, Q., and Xiong, H. (2020). Latency-Aware Differentiable Neural Architecture Search. *CoRR*, abs/2001.06392.

Yang, A., Esperança, P. M., and Carlucci, F. M. (2019). Nas evaluation is frustratingly hard. *arXiv preprint arXiv:1912.12522*.

Yang, T., Howard, A. G., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. (2018). NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part X*, volume 11214 of *Lecture Notes in Computer Science*, pages 289–304. Springer.

Yao, S., Zhao, Y., Shao, H., Liu, S., Liu, D., Su, L., and Abdelzaher, T. (2018). FastDeepIoT: Towards Understanding and Optimizing Neural Network Execution Time on Mobile and Embedded Devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 278–291.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. (2019). NAS-Bench-101: Towards Reproducible Neural Architecture Search. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114. PMLR.

Yu, K., Ranftl, R., and Salzmann, M. (2020). How to Train Your Super-Net: An Analysis of Training Heuristics in Weight-Sharing NAS. *CoRR*, abs/2003.04276.

Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018). ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 6848–6856. IEEE Computer Society.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.

A Best practices for NAS, Code and Data

To improve the reproducibility and facilitate fair experimental comparisons, we follow the best-practices checklist (Lindauer and Hutter, 2020):

- **Release Code for the Training Pipeline(s) you use**. Our experiments are based on White et al. (2021), who use NASLib (Ruchte et al., 2020) to compare 31 methods for accuracy prediction. Our NASLib fork, extending the framework for HW-NAS-Bench, TransNAS-Bench, some performance predictors and the hypervolume simulations, is provided in the supplementary materials. We intend to either make our fork available on GitHub or submit the changes upstream once this paper is accepted/published.

  It is noteworthy that we chose to explicitly normalize input data values (e.g. latency or energy consumption) for all predictors and datasets, which reduces the dependency of hyper-parameters (e.g. learning rate) on the dataset and allows us to analyze and compare the prediction errors across datasets more effectively.

- **Use the Same Evaluation Protocol for the Methods Being Compared**. Aside from the implementation of each predictor, all experiments use the same pipeline.

- **Validate The Results Several Times**. We ran each predictor 50 times, with seeds $\{0, ..., 49\}$. The reductions in hypervolume are simulated 1000 times using different a different subset of the data set, for each combination of {iteration, HW-NAS data set, noise on HW metric}.

- **Control Confounding Factors**. While all experiments used the same software libraries and hardware resources, they were run on different machines to speed up the evaluation. We found hardly any benefit in using a GPU even for the network-based predictors, which is why every method only used two CPU cores. Single experiments vary in time, usually ranging from few minutes to an hour. The OS is Ubuntu 18.04, notable software packages are PyTorch 1.9.0, numpy 1.19.5, scikit-learn 0.24.2, pybnn 0.0.5, ngboost 0.3.11, and xgboost 1.4.2

- **Report the Use of Hyperparameter Optimization**. See Appendix C.

In addition to the code in the supplementary materials, we also provide the experimental results as csv files. Running the predictors and hypervolume simulations takes some time, but the easy access to the data of the finished experiments may prove useful for future research. Please see *readme.md* in the accompanying code zip file for instructions.

We used the following data/assets/code:

- HW-NAS-Bench: `https://github.com/RICE-EIC/HW-NAS-Bench`, MIT License.

- TransNAS-Bench-101: `https://github.com/kmdanielduan/TransNASBench`, the repository and website lack an explicit license statement.

- NASLib: `https://github.com/automl/NASLib`, Apache License

B Encodings and Predictors

B.1 Data encodings

Every architecture $a \in \mathcal{A}$ requires a unique representation, which depends on the used predictor. The common encoding types are:

Adjacency one-hot: Each architecture a is uniquely defined by the chosen candidate operation on every path. For example, each architecture in NAS-BENCH-201 consists of a repeated cell structure, which has five candidate operations on each of the six paths. Therefore there are

$5^6 = 15625$ unique architectures, which can each be referenced by a sequence of operation-indices such as [0 1 2 3 4 0]. Many predictors perform better if the sequence is presented as a one-hot encoding, which is in this case [10000 01000 00100 00010 00001 10000].

Similarly, the **path-encoding** (used by BANANAS) is a one-hot representation over the used candidate operation all possible paths. Since the connectivity within cells for HW-NAS-Bench and TransNAS-Bench-101 is fixed, it provides no more information than the adjacency one-hot encoding. If the connectivity can be adjusted more freely, as in the NAS-Bench-101 search space, the additional information may improve the fit.

The encodings for **BONAS**, **GCN**, and **NAO** each provide further information in addition to the Adjacency one-hot vector, most notably the adjacency-matrix. This $\{0,1\}^{(N+2)\times(N+2)}$ matrix lists describes which of the N architecture paths (rows) serves as inputs for each other path (column), and also includes input/output.

B.2 Predictors

We briefly describe the 18 predictor methods in our experiments. We adopt their implementations from the NASLib library (see Appendix A), which we extend with Linear Regression, Ridge Regression, and Support Vector Machines from the scikit-learn package; and a simple Lookup Table implementation. Unless specified otherwise, the methods use the adjacency one-hot encoding.

- **BANANAS** An ensemble of three MLP models with five to 20 layers, each using the path-encoding (White et al., 2019).
- **Bayesian Linear Regression** A bayesian model that assumes (1) a linear dependency between inputs and outputs, and (2) that the samples are normally distributed (Bishop, 2007).
- **BOHAMIANN** A bayesian inference predictor using stochastic gradient Hamiltonian Monte Carlo (SGHMC) to sample from a bayesian neural network (Springenberg et al., 2016).
- **BONAS** Bayesian Optimization for NAS (Shi et al., 2020) uses a GCN predictor within an outer loop of bayesian optimization, as a meta-learning task. The GCN requires encoding the adjacency matrix of each architecture.
- **Gaussian Process** A simple model that assumes a joint Gaussian distribution underlying the training data (Rasmussen, 2003).
- **GCN** A Graph Convolutional Network that makes use of an adjacency-matrix encoding of each architecture (Wen et al., 2020).
- **Linear Regression** A simple model that assumes an independent value/cost for each operation/layer, which only need to be summed up. Unlike the Lookup Table model, it uses a least-square fit on the training data.
- **Lookup Table** The most simple and perhaps widely used model for differentiable architecture selection. It generally assumes a single baseline architecture (e.g. [001 001] in adjacency one-hot encoding), and a lookup matrix $\mathbb{R}^{(\text{num layers})\times(\text{num candidates})}$ that contains the increases/reductions in the metric for each layer and candidate operation. The metric value of a new architecture can be predicted with a simple sum over the respective matrix entries and the baseline value. The model is obtained from measuring either each candidate operation in isolation, or by computing the differences between the baseline architecture and specific variations (e.g. [010 001] or [100 001], to measure the first candidates). This model always requires $1+(\textit{num layers}) \cdot (\textit{num candidates}-1)$ neighbored architectures to fit. We detail the resulting correlation values for each used dataset in Appendix E.

- **LGBoost** Light Gradient Boosting Machine (LightGBM or LGBoost, Ke et al. (2017)) is a lightweight gradient-boosted decision tree model.
- **MLP** We use fully-connected Multi Layer Perceptrons in two size-categories.
- **NAO** (Luo et al., 2018) uses an encoder-decoder topology, which encodes/compresses an architecture to a continuous representation, and decodes it again. This representation is further used to make architecture predictions.
- **NGBoost** Natural Gradient Boosting (NGBoost, Duan et al. (2020)) is a gradient-boosted decision tree model that uses natural gradients to estimate uncertainty.
- **Ridge Regression** Ridge Regression (Saunders et al., 1998) extends the Linear Regression least-squares fit with a regularization term that serves as bias-variance tradeoff.
- **Random Forests** An ensemble of decision trees (Liaw et al., 2002).
- **Sparse Gaussian Process** an approximation of Gaussian Processes that summarizes training data (Candela and Rasmussen, 2005).
- **Support Vector Machine** A model that maps its inputs to a high-dimensional space, where training samples are used as support-vectors for decision-boundaries (Cortes and Vapnik, 1995).
- **XGBoost** eXtreme Gradient Boosting (XGBoost, Chen and Guestrin (2016)) is a gradient-boosted decision tree model.

C Hyperparameters

The default hyperparameters of the used predictors vary significantly in their levels of hyper-parameter tuning, especially in the context of NAS. Additionally, some predictors may internally make use of cross-validation, while others do not. Following White et al. (2021), we attempt to level the playing field by running a cross-validation random-search over hyper-parameters each time a predictor is fit to data. Each search is limited to 5000 iterations and a total run time of 15 minutes and naturally excludes any test data.

We list our default and hyper-parameter sample ranges in Table 2. For comparability with White et al. (2021), we only change the values of newly introduced parameterized predictors: Ridge Regression, Support Vector Machines, and small MLPs.

Table 2: Hyper-parameter ranges and default values of the configurable predictors

Model	Hyper-parameter	Range/Choice	Log-transform	Default
BANANAS	Num. Layers	[5, 25]	false	20
	Layer width	[5, 25]	false	20
	Learning rate	[0.0001, 0.1]	true	0.001
BONAS	Num. Layers	[16, 128]	true	64
	Batch size	[32, 256]	true	128
	Learning rate	[0.00001, 0.1]	true	0.0001
GCN	Num. Layers	[64, 200]	true	144
	Batch size	[5, 32]	true	7
	Learning rate	[0.00001, 0.1]	true	0.0001
	Weight decay	[0.00001, 0.1]	true	0.0003
LGBoost	Num. leaves	[10, 100]	false	31
	Learning rate	[0.001, 0.1]	true	0.05
	Feature fraction	[0.1, 1]	false	0.9
MLP (small)	Num. layers	[2, 5]	false	3
	Layer width	[16, 128]	true	32
	Learning rate	[0.0001, 0.1]	true	0.001
	Activation function	{relu, tanh, hardswish}		relu
MLP (huge)	Num. layers	[5, 25]	false	20
	Layer width	[5, 25]	false	20
	Learning rate	[0.0001, 0.1]	true	0.001
NAO	Num. layers	[16, 128]	true	64
	Batch size	[32, 256]	true	100
	Learning rate	[0.00001, 0.1]	true	0.001
NGBoost	Num. estimators	[128, 512]	true	64
	Learning rate	[0.001, 0.1]	true	0.081
	Max depth	[1, 25]	false	6
	Max features	[0.1, 1]	false	0.79
Ridge Regression	Regularization α	[0.25, 2.5]	false	1.0
Random Forests	Num. estimators	[16, 128]	true	116
	Max features	[0.1, 0.9]	true	0.17
	Min samples (leaf)	[1, 20]	false	2
	Min samples (split)	[2, 20]	true	2
Support Vector Machine	Regularization C	[0.5, 1.5]	false	1.0
	Kernel	{linear, poly, rbf, sigmoid}		rbf
XGBoost	Max depth	[1, 15]	false	6
	Min child weight	[1, 10]	false	1
	Col sample (tree)	[0, 1]	false	1
	Learning rate	[0.001, 0.5]	true	0.3
	Col sample (level)	[0, 1]	false	1

D NAS-Bench-201 / HW-NAS-Bench cell design

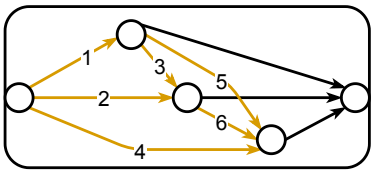


Figure 5: Basic NAS-Bench-201 / HW-NAS cell design. Each of the six orange paths is finalized with exactly one out of five candidate operations {Zero, Skip, Convolution 1×1, Convolution 3×3, Average Pooling 3×3}.

E Selection of datasets

Table 3: Kendall's Tau test correlation for Linear Regression, XGBoost, and Lookup Tables (LUT) on all HW-NAS-Bench datasets (rows), for different amounts of available training data (columns), tested on the remaining 625 samples. The Lookup Table model is tested on all 15625 architectures. We selected the five data sets at the top.

	Linear Regression										XGBoost	LUT
	11	25	55	124	276	614	1366	3036	6748	15000	15000	-
ImageNet16-120-raspi4_latency	0.324	0.205	0.606	0.676	0.705	0.716	0.715	0.723	0.728	0.729	0.757	0.443
cifar100-pixel3_latency	0.392	0.292	0.732	0.780	0.797	0.803	0.806	0.809	0.812	0.812	0.877	0.484
cifar10-edgegpu_latency	0.370	0.258	0.724	0.790	0.806	0.819	0.820	0.822	0.830	0.829	0.926	0.175
cifar100-edgegpu_energy	0.376	0.275	0.732	0.793	0.812	0.821	0.821	0.823	0.831	0.831	0.920	0.221
ImageNet16-120-eyeriss arith. int.	0.369	0.293	0.748	0.805	0.817	0.827	0.825	0.832	0.843	0.846	0.970	0.861
cifar10-pixel3_latency	0.388	0.300	0.733	0.780	0.797	0.805	0.805	0.810	0.813	0.813	0.878	0.475
cifar10-raspi4_latency	0.393	0.315	0.740	0.787	0.799	0.805	0.807	0.810	0.813	0.813	0.890	0.462
cifar100-raspi4_latency	0.393	0.308	0.744	0.786	0.801	0.807	0.810	0.810	0.814	0.814	0.888	0.445
ImageNet16-120-pixel3_latency	0.398	0.312	0.739	0.786	0.799	0.807	0.809	0.812	0.815	0.816	0.884	0.509
cifar100-edgegpu_latency	0.375	0.268	0.728	0.793	0.810	0.821	0.820	0.822	0.831	0.831	0.924	0.191
cifar10-edgegpu_energy	0.375	0.284	0.728	0.792	0.810	0.821	0.823	0.824	0.831	0.831	0.922	0.183
ImageNet16-120-edgegpu_energy	0.377	0.281	0.733	0.797	0.814	0.825	0.825	0.826	0.834	0.833	0.926	0.280
ImageNet16-120-edgegpu_latency	0.379	0.264	0.737	0.799	0.817	0.826	0.826	0.828	0.836	0.835	0.938	0.277
cifar10-eyeriss arith. int.	0.384	0.296	0.757	0.811	0.826	0.835	0.832	0.843	0.854	0.854	0.969	0.826
cifar100-eyeriss arith. int.	0.384	0.297	0.757	0.811	0.826	0.835	0.833	0.844	0.855	0.856	0.971	0.830
ImageNet16-120-fpga_latency	0.443	0.494	0.904	0.936	0.947	0.951	0.948	0.951	0.952	0.952	0.983	0.965
ImageNet16-120-fpga_energy	0.443	0.494	0.905	0.935	0.947	0.951	0.948	0.951	0.952	0.952	0.983	0.965
ImageNet16-120-eyeriss_latency	0.457	0.937	0.953	0.954	0.954	0.954	0.953	0.953	0.954	0.954	0.952	0.989
cifar10-eyeriss_latency	0.461	0.943	0.959	0.959	0.960	0.960	0.959	0.960	0.960	0.960	0.958	0.995
cifar100-eyeriss_latency	0.462	0.946	0.963	0.963	0.963	0.963	0.963	0.963	0.964	0.963	0.962	0.998
cifar10-eyeriss_energy	0.456	0.967	0.985	0.985	0.985	0.985	0.985	0.985	0.985	0.985	0.975	0.996
ImageNet16-120-eyeriss_energy	0.458	0.967	0.985	0.985	0.986	0.985	0.986	0.985	0.985	0.986	0.972	0.998
cifar100-eyeriss_energy	0.457	0.967	0.985	0.985	0.985	0.986	0.985	0.986	0.986	0.986	0.976	0.998
cifar10-fpga_energy	0.458	0.973	0.987	0.987	0.987	0.987	0.987	0.987	0.987	0.987	0.986	0.999
cifar100-fpga_energy	0.458	0.973	0.987	0.987	0.987	0.987	0.987	0.987	0.987	0.987	0.986	0.999
cifar100-fpga_latency	0.457	0.973	0.987	0.987	0.987	0.987	0.987	0.987	0.987	0.987	0.986	0.999
cifar10-fpga_latency	0.457	0.973	0.987	0.987	0.987	0.987	0.987	0.987	0.987	0.987	0.986	0.999

Table 4: Kendall's Tau test correlation for Linear Regression and XGBoost on the five used TransNAS datasets (rows), for different amounts of available training data (columns), tested on the remaining 256 samples. The Lookup Table model (LUT) is tested on all 3256 architectures.

	Linear Regression										XGBoost	LUT
	9	18	34	65	123	234	442	837	1585	2999	2999	-
jigsaw	0.201	0.227	0.410	0.535	0.586	0.605	0.616	0.624	0.631	0.632	0.661	0.201
class_object	0.268	0.262	0.518	0.646	0.711	0.741	0.759	0.771	0.780	0.780	0.828	0.701
room_layout	0.275	0.271	0.527	0.653	0.721	0.753	0.768	0.780	0.789	0.789	0.896	0.685
class_scene	0.275	0.268	0.527	0.653	0.721	0.755	0.768	0.782	0.789	0.790	0.907	0.710
segmentsemantic	0.282	0.259	0.545	0.684	0.746	0.780	0.798	0.809	0.816	0.818	0.871	0.726

HW-NAS-Bench:. To select five datasets that are (1) non-linear and (2) different from one another, we first fit Linear Regression to every available dataset, with the results listed in Table 3. The bottom 12 datasets can be accurately fit with only 25 training samples, so they are not very interesting as a challenge. On these datasets, the Lookup Table model achieves exceptional performance. Since the networks for CIFAR10, CIFAR100 and ImageNet16-120 only differ slightly, their measurements on the same device and metric (e.g. raspi4 latency) is very similar. To improve the generalizability of our results, we thus select datasets on different devices and metrics, which are listed at the top of Table 3. As displayed in Figure 6, their data distributions are generally different.

TransNAS-Bench-101:. As shown in Figure 7, the latency measurements of the architectures is generally very similarly distributed. We evaluate the possibly redundant datasets nonetheless, since latency predictions in macro-level search spaces are an important domain for NAS on image classification and object detection tasks. We select all data sets that provide the *test_loss* and *inference_time* attributes for all architectures, resulting in exactly the five datasets listed in Section 4 (the other two datasets contain more specific test losses).

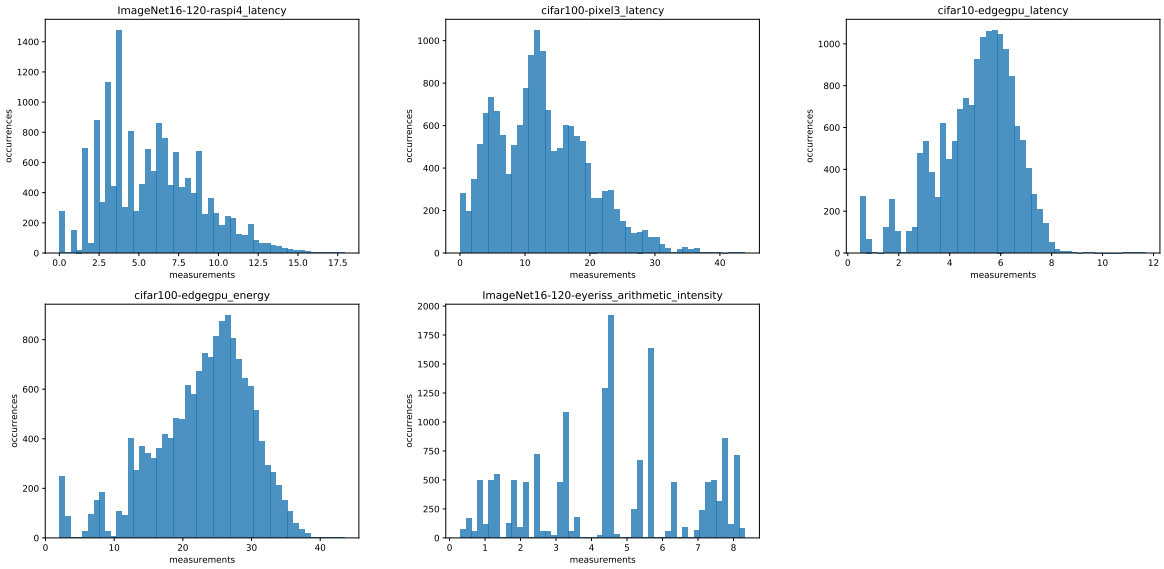


Figure 6: How the data of each selected HW-NAS-Bench dataset is distributed (not normalized).

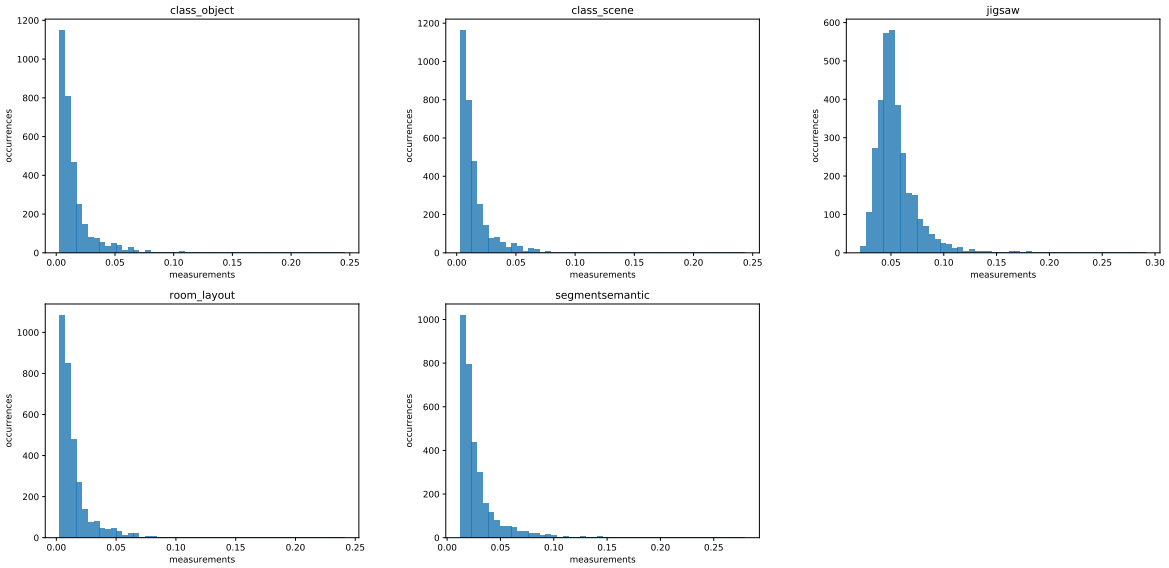


Figure 7: How the data of each selected TransNAS-Bench-101 dataset is distributed (not normalized). Since all architectures are measured for latency on the same hardware, the resulting datasets are much less diverse than the HW-NAS-Bench ones.

F Predictor fit time

We visualize the average fit time of prediction models in Figure 8. Note that this may include a brief hyper-parameter optimization of up to 15minutes.

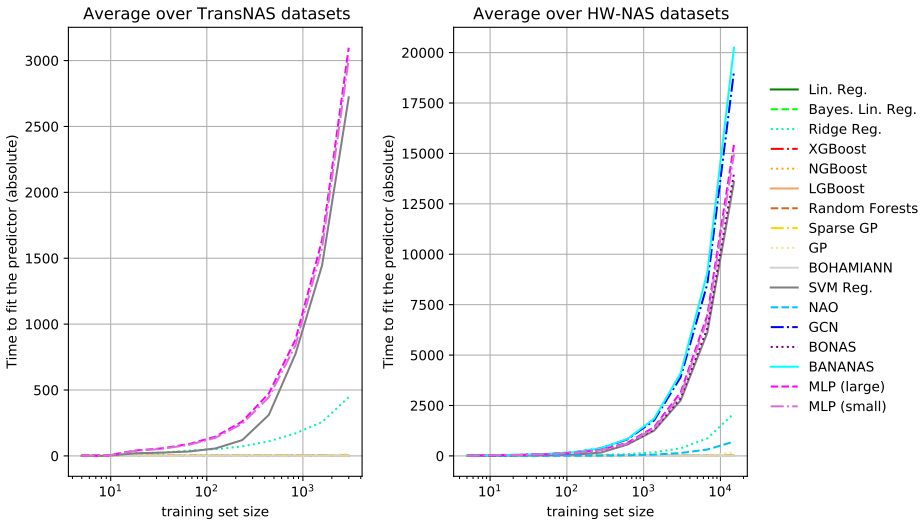


Figure 8: Fit time (in seconds) of predictors to data, depending on the training set size. By far the most expensive methods are network-based. However, a significant portion of this time is spent on the hyper-parameter optimization prior to the actual fitting.

G Approximating predictor mistakes

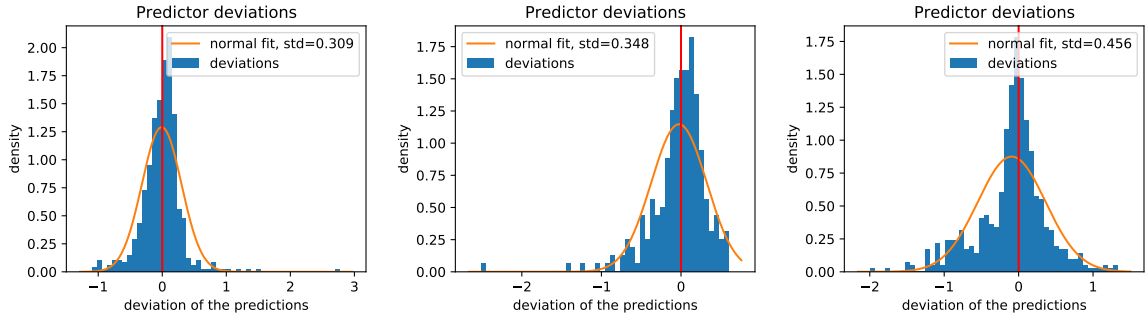


Figure 9: Further examples of predictor deviation distributions, as visualized in the center of Figure 2. **Left**: Linear Regression on CIFAR100, edgegpu, energy consumption. **Center**: Support Vector Machine on Jigsaw. **Right**: small MLP on ImageNet16-120, raspi4, latency.

Intuitively, the predictor deviation distributions (see Figures 2 and 9) generally resemble a normal distribution. However, most predictors:

(1) Have a notable peak, sometimes off-center (e.g. at x=0.2)

(2) Have less density than a normal distribution almost everywhere else

(3) Have some outliers (e.g. at x>1.5) that are extremely unlikely for a normal distribution

Table 5: P-values of different distributions, trying to fit the distribution of all predictor mistakes according to a t-test. Larger values are better, but comparing many empirically sampled points with a true density function tends to push the p-values to 0.

	p-value
normal	0.028
cauchy	0.030
lognorm	0.028
t	0.028
uniform	0.037

We measured the p-value for different distributions on the first 100 test samples using a T-Test, every time we evaluated a predictor. The average statistics can be found in Table 5. Since a large number of empirical observations generally pushes the p-value to 0, this only serves to compare them to each other. We find that the outliers (3) appear often enough and are so unlikely to happen for a normal distribution, that even a uniform distribution has a higher statistical support. Consequentially, we approximate the common predictor deviations by sampling from a mixed distribution that adresses (1) to (3).

This mixed distribution consists of two Normal distributions (N_1, N_2) and one Uniform distribution (U), from which we sample with 72.5%, 26.5% and 1% respectively. For some constant v:

- We uniformly sample a shift c from $[0, 2 \cdot v]$, that is used to push the centers of N_1 and N_2 to $x > 0$ and $x < 0$ respectively.

- We sample each value from $N_1(c, v)$, $N_2(-c, 3 \cdot v)$, and $U_1(-15 \cdot v, 15 \cdot v)$ randomly, with the weighting given above.
- We normalize (subtract mean, divide by standard deviation) our sampled distribution and then scale it to the desired standard deviation.
- The predictors produce non-smooth distributions. We simulate that by sampling 15 times fewer values as needed, and repeat them as often.

The code for the simulation is also provided (see Appendix A). As seen in Figure 10, the resulting simulated deviation distributions generally resemble a common predictor pattern. We do not account for differences in predictors, training set sizes or more, since that may become too specific and over-engineered.

Appendix I visualizes simulation sanity checks. We find that the simulation is slightly pessimistic and simplified, but resembles the results of actual predictors.

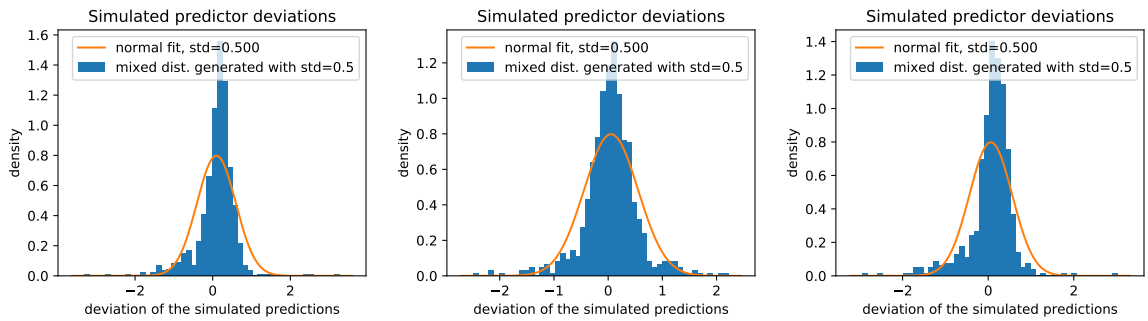


Figure 10: The sampled values of gaussian+uniform fit the measured predictor mistakes better than a single distribution, as they are roughly normally distributed, but include outliers.

H Measuring simulated mistakes

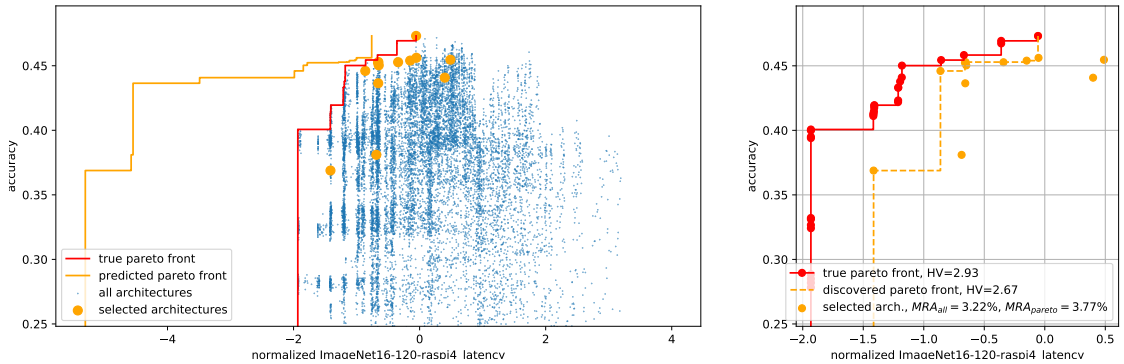


Figure 11: Similar to Figure 3. When the discovered Pareto set is considerably worse than the true Pareto set, it is possible for the Mean Reduction of Accuracy of the Pareto subset (MRA_{pareto}) to be *worse* than the average over all architectures (MRA_{all}). This naturally happens more frequently for worse predictors with a high sampling std. and low KT correlation. Consequentially, the difference between MRA_{all} and MRA_{pareto} is wider for better predictors (see Figure 4). Additionally, all of the selected non-Pareto-front members are clustered in a high-latency area and redundant with each other. This emphasizes the limitations of just considering drops in accuracy, as the hardware metric aspect is ignored. In this case, the predictor-guided selection failed to find a low-latency solution. Hypervolume solves these problems but is a less intuitive metric.

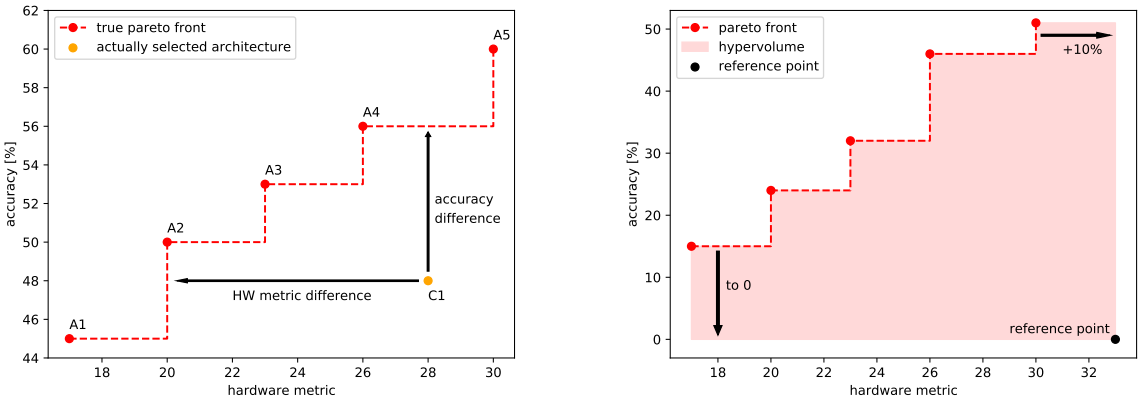


Figure 12: Examples to explain measurement methods.
Left: The distance of each selected candidate architecture C1 to the true Pareto front is measured, for accuracy and the hardware metric. C1 is dominated by A2, A3, and A4 of the true Pareto set. A2 has a slightly higher accuracy than C1 while being much better on the hardware metric, e.g. latency. A4 has a slightly better hardware metric value, but much higher accuracy. Given several candidate architectures, their differences are averaged.
Right: We compute the reference point for the hypervolume (for two objectives: area under a Pareto front) by multiplying the highest hardware metric value from the true Pareto front with 1.1, and accuracy 0. While we are consistent throughout all experiments, this choice is arbitrary, as there is no obviously correct choice for the reference point. If the hypervolume of a supposed Pareto front is computed, the reference point of the true Pareto front is reused. Thus, choosing inferior architectures will always reduce the hypervolume. We arbitrarily chose the multiplier of $m = 1.1$ as a middle ground between making the rightmost point of the Pareto front irrelevant ($m = 1.0$) and overemphasizing it ($m >> 1.0$).

I Simulation sanity check

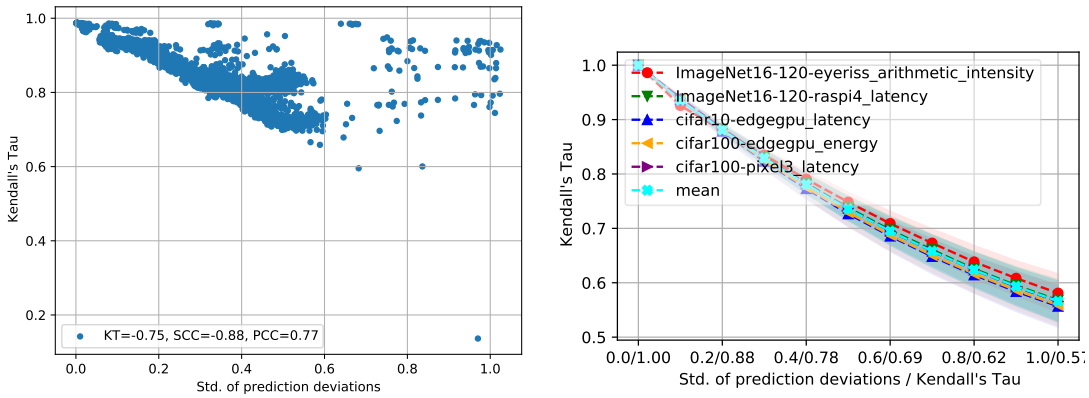


Figure 13: Standard deviation over the predictor deviations (x axis) and Kendall's Tau correlation (y axis), for the trained predictors on HW-NAS-Bench (left) and in simulation (right). The simulated predictor inaccuracies are slightly pessimistic (low KT), but still match the true values.

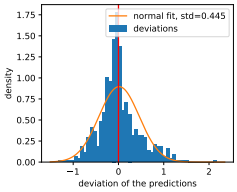
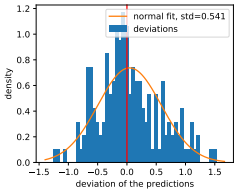
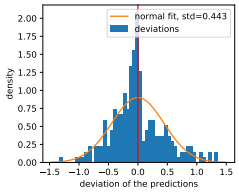
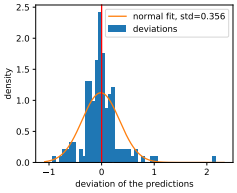
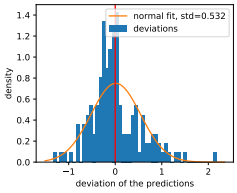
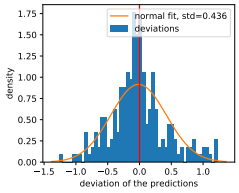
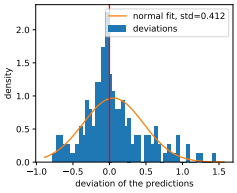
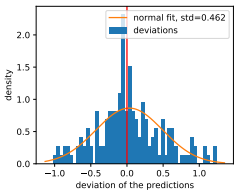
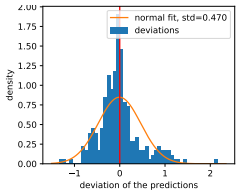
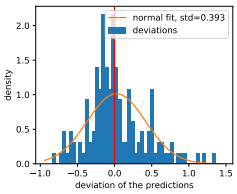
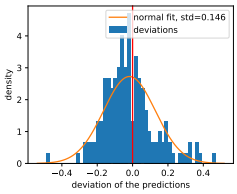
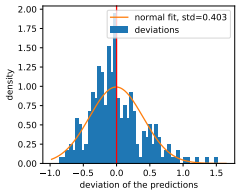
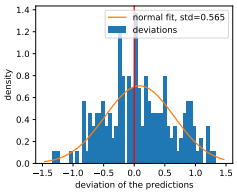
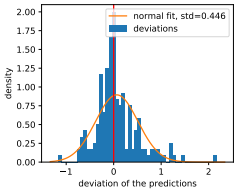
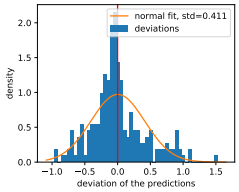
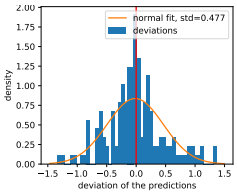
All candidates			
	candidate occurrences in the architecture		
candidate	not at all	exactly once	exactly twice
Zero			
Skip			
Conv1x1			
Conv3x3			
Pool			

Table 6: How a trained XGB predictor deviates from the ground-truth values for different architecture subsets, akin to Figure 2. While they are not exactly the same, they still resemble the distribution over the entire test set (top plot, 625 samples). One noteworthy exception is when no Conv3x3 operations are used at all, in which case the standard deviation is considerably smaller.