A Related Work

A.1 Existing NAS benchmarks

Benchmarks for NAS were introduced only recently with NAS-Bench-101 (Ying et al., 2019) as the first among them. NAS-Bench-101 is a tabular benchmark consisting of $\sim$423k unique architectures in a cell structured search space evaluated on CIFAR-10 (Krizhevsky, 2009). To restrict the number of architectures in the search space, the number of nodes and edges was given an upper bound and only three operations are considered. One result of this limitation is that One-Shot NAS methods can only be applied to subspaces of NAS-Bench-101 as demonstrated in NAS-Bench-1Shot1 (Zela et al., 2020b).

NAS-Bench-201 (Dong & Yang, 2020), in contrast, uses a search space with a fixed number of nodes and edges, hence allowing for a straight-forward application of one-shot NAS methods. However, this limits the total number of unique architectures to as few as 6466. NAS-Bench-201 includes evaluations of all these architectures on three different datasets, namely CIFAR-10, CIFAR-100 (Krizhevsky, 2009) and Downsampled Imagenet 16×16 (Chrabaszcz et al., 2017), allowing for transfer learning experiments.

NAS-Bench-NLP (Klyuchnikov et al., 2020) was recently proposed as a tabular benchmark for NAS in the Natural Language Processing domain. The search space resembles NAS-Bench-101 as it limits the number of edges and nodes to constrain the search space size resulting in 14k evaluated architectures.

A.2 Neural Network Performance Prediction

In the past, several works have attempted to predict the performance of neural networks by extrapolating learning curves (Domhan et al., 2015; Klein et al., 2017; Baker et al., 2017). A more recent line of work in performance prediction focuses more on feature encoding of neural architectures. Peephole (Deng et al., 2017) and TAPAS (Istrate et al., 2019) both use an LSTM to aggregate information about the operations in chain-structured architectures. On the other hand, BANANAS (White et al., 2019) introduces a path-based encoding of cells that automatically resolves the computational equivalence of architectures.

Graph Neural Networks (GNNs) (Gori et al., 2005; Kipf & Welling, 2017; Zhou et al., 2018; Wu et al., 2019) with their capability of learning representations of graph-structured data appear to be a natural choice to learning embeddings of NN architectures. Shi et al. (2019) and Wen et al. (2019) trained a Graph Convolutional Network (GCN) on a subset of NAS-Bench-101 (Ying et al., 2019) showing its effectiveness in predicting the performance of unseen architectures. Moreover, Friede et al. (2019) propose a new variational-sequential graph autoencoder (VS-GAE) which utilizes a GNN encoder-decoder model in the space of architectures and generates valid graphs in the learned latent space.

Several recent works further adapt the GNN message passing to embed architecture bias via extra weights to simulate the operations such as in GATES (Ning et al., 2020) or integrate additional information on the operations (e.g. flop count) (Xu et al., 2019b). Tang et al. (2020) chose to operate GNNs on relation graphs based on architecture embeddings in a metric learning setting, allowing to pose NAS performance prediction as a semi-supervised setting.

B Training Details for the GIN in the Motivation

We set the GIN to have a hidden dimension of 64 with 4 hidden layers resulting in around $\sim$40k parameters. We trained for 30 epochs with a batch size of 128. We chose the MSE loss function and add a logarithmic transformation to emphasize the data fit on well-performing architectures.

C NAS-Bench-301 Dataset

C.1 Search Space

We use the same architecture search space as in DARTS (Liu et al., 2019). Specifically, the normal and reduction cell each consist of a DAG with 2 input nodes (receiving the output feature maps from

the previous and previous-previous cell), 4 intermediate nodes (each adding element-wise feature maps from two previous nodes in the cell) and 1 output node (concatenating the outputs of all intermediate nodes). Input and intermediate nodes are connected by directed edges representing one of the following operations: {Sep. conv 3×3, Sep. conv 5×5, Dil. conv 3×3, Dil. conv 5×5, Max pooling 3×3, Avg. pooling 3×3, Skip connection}.

C.2 Data Collection

To achieve good global coverage, we use random search to evaluate ∼23k architectures. We note that space-filling designs such as quasi-random sequences, e.g. Sobol sequences (Sobol', 1967), or Latin Hypercubes (McKay et al., 2000) and Adaptive Submodularity (Golovin & Krause, 2011) may also provide good initial coverage.

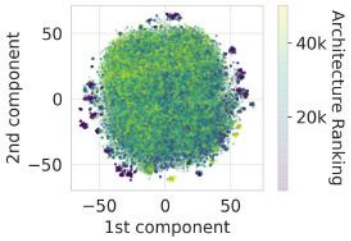


Figure 6: t-SNE visualization of the sampled architectures.

Random search is supplemented by data which we collect from running a variety of optimizers, representing Bayesian Optimization (BO), evolutionary algorithms and One-Shot Optimizers. We used Tree-of-Parzen-Estimators (TPE) (Bergstra et al., 2011) as implemented by Falkner et al. (2018) as a baseline BO method. Since several recent works have proposed to apply BO over combinatorial spaces (Oh et al., 2019; Baptista & Poloczek, 2018) we also used COMBO (Oh et al., 2019). We included BANANAS (White et al., 2019) as our third BO method, which uses a neural network with a path-based encoding as a surrogate model and hence scales better with the number of function evaluations. As two representatives of evolutionary approaches to NAS, we chose Regularized Evolution (RE) (Real et al., 2019) as it is still one of the state-of-the art methods in discrete NAS and Differential Evolution (Price et al., 2006) as implemented by Awad et al. (2020). Accounting for the surge in interest in One-Shot NAS, our collected data collection also entails evaluation of architectures from search trajectories of DARTS (Liu et al., 2019), GDAS (Dong & Yang, 2019), DrNAS (Chen et al., 2020) and PC-DARTS (Xu et al., 2020). For details on the architecture training details, we refer to Section C.6.

For each architecture $a \in \mathcal{A}$, the dataset contains the following metrics: train/validation/test accuracy, training time and number of model parameters.

C.3 Optimizer Performance

The trajectories from the different NAS optimizers yield quite different performance distributions. This can be seen in Figure 7 which shows the ECDF of the validation errors of the architectures evaluated by each optimizer. As the computational budgets allocated to each optimizer vary widely, this data does not allow for a fair comparison between the optimizers. However, it is worth mentioning that the evaluations of BANANAS feature the best distribution of architecture performances, followed by PC-DARTS, DrNAS, DE, GDAS, and RE. TPE only evaluated marginally better architectures than RS, while COMBO and DARTS evaluated the worst architectures.

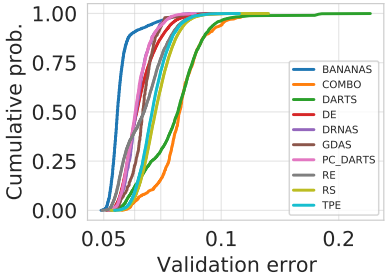


Figure 7: Empirical Cumulative Density Function (ECDF) plot comparing all optimizers in the dataset. Optimizers which cover good regions of the search space feature higher values in the low validation error region.

In Figure 6 we visualize the overall coverage of the search space as well as the similarity between sampled architectures using t-SNE (van der Maaten & Hinton, 2008). Besides showing a good overall coverage, some well-performing architectures in the search space form distinct clusters which are mostly located outside the main cloud of points. This clearly indicates that architectures with similar performance are close to each other in the architecture space. Additionally, we observe that different optimizers sample different types of architectures, see Figure 8.

We also perform a t-SNE analysis on the data collected by the different optimizers in Figure 8. We find that RE discovers well-performing architectures which form clusters distinct from the architectures found via RS. We observe that COMBO searched previously unexplored areas of the search

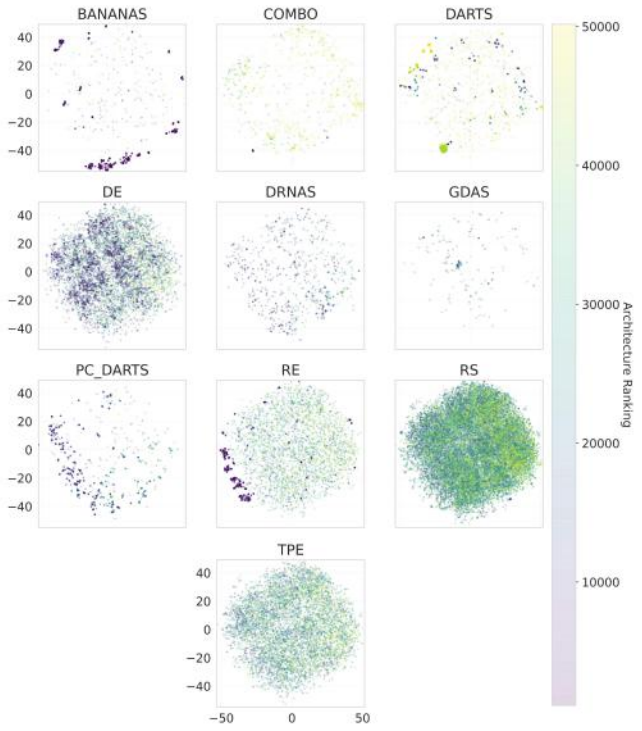


Figure 8: Visualization of the exploration of different parts of the architectural t-SNE embedding space for all optimizers used for data collection. The architecture ranking by validation accuracy (lower is better) is global over the entire data collection of all optimizers.

space. BANANAS, which found some of the best architectures, explores clusters outside the main cluster. However, it heavily exploits regions at the cost of exploration. We argue that this is a result of the optimization of the acquisition function via random mutations based on the previously found iterates, rather than on new random architectures. DE is the only optimizer which finds well performing architectures in the center of the embedding space.

C.4 Cell topology, operations and noise

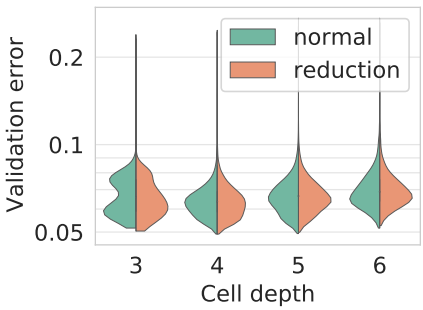


Figure 9: Distribution of the validation error for different cell depth.

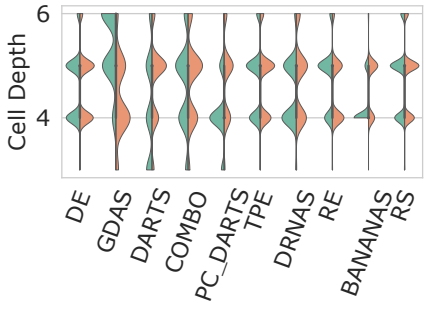


Figure 10: Comparison between the normal and reduction cell depth for the architectures found by each optimizer.

In this section, we investigate the influence of the cell topology and the operations on the performance of the architectures in our setting. The discovered properties of the search space inform our choice of metrics for the evaluation of different surrogate models.

First, we study how the validation error depends on the depth of architectures. Figure 9 visualizes the performance distribution of normal and reduction cells of different depth[5] by approximating empirical distributions with a kernel density estimation used in violin plots (Hwang et al., 1994). We observe that the performance distributions are similar for the normal and reduction cells with the same cell depth. Although cells of all depths can reach high performances, shallower cells seem slightly favored. Note that these observations are subject to changes in the hyperparameter setting, e.g. training for more epochs may render deeper cells more competitive. The best-found architecture features a normal and reduction cell of depth 4. Color-coding the cell depth in our t-SNE projection also confirms that the t-SNE analysis captures the cell depth well as a structural property (c.f. Figure 12). It also reinforces that the search space is well-covered.

We also show the distribution of normal and reduction cell depths of each optimizer in Figure 10 to get a sense for the diversity between the discovered architectures. We observe that DARTS and BANANAS generally find architectures with a shallow reduction cell and a deeper normal cell, while the reverse is true for RE. DE, TPE, COMBO and RS appear to find normal and reduction cells with similar cell depth.

Aside from the cell topology, we can also use our dataset to study the influence of operations to the architecture performance. The DARTS search space contains operation choices without parameters such as Skip-Connection, Max Pooling 3×3 and Avg Pooling 3×3. We visualize the influence of these parameter-free operations on the validation error in the normal and reduction cell in Figure 16a, respectively Figure 13. While pooling operations in the normal cell seem to have a negative impact on performance, a small number of skip connections improves the overall performance. This is somewhat expected, since the normal cell is dimension preserving and skip connections help training by improving gradient flow like in ResNets (He et al., 2016). In the reduction cell, the number of parameter-free operations has less effect as shown in Figure 13. In contrast to the normal cell where 2-3 skip-connections lead to generally better performance, the reduction cell shows no similar trend. For both cells, however, featuring many parameter-free operations significantly deteriorates performance. We therefore expect that a good surrogate also models this case as a poorly performing region.

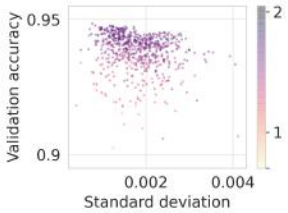


Figure 11: Standard deviation of the val. accuracy for multiple architecture evaluations.

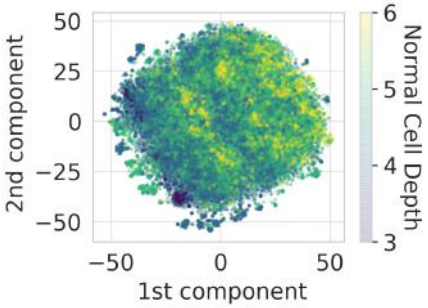


Figure 12: t-SNE projection colored by the depth of the normal cell.

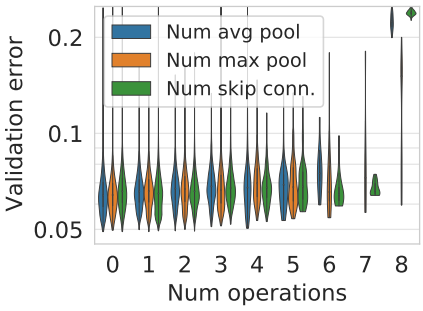


Figure 13: Distribution of validation error in dependence of the number of parameter-free operations in the reduction cell. Violin plots are cut off at the respective observed minimum and maximum value.

C.5 Noise in Architecture Evaluations

As discussed in Section 2, the noise in architecture evaluations can be large enough for surrogate models to yield more realistic estimates of architecture performance than a tabular benchmark based

[5]We follow the definition of cell depth used by Shu et al. (2020), i.e. the length of the longest simple path through the cell.

on a single evaluation per architecture. To study the magnitude of this noise on NAS-Bench-301, we evaluated 500 architectures randomly sampled from our Differential Evolution (DE) (Awad et al., 2020) run with 5 different seeds each.[6] We find a mean standard deviation of $1.6\text{e}{-3}$ for the final validation accuracy which is slightly less than the noise observed in NAS-Bench-101 (Ying et al., 2019); one possible reason for this could be a more robust training pipeline. Figure 11 in the Appendix shows that, while the noise tends to be lower for the best architectures, a correct ranking based on a single evaluation is still difficult. Finally, we compare the MAE when estimating the architecture performance from only one sample to the results from Table 1. Here, we also find a slightly lower MAE of $1.38\text{e}{-3}$ than for NAS-Bench-101.

C.6 Training details

Each architecture was evaluated on CIFAR-10 (Krizhevsky, 2009) using the standard 40k, 10k, 10k split for train, validation and test set. The networks were trained using SGD with momentum 0.9, initial learning rate of 0.025 and a cosine annealing schedule (Loshchilov & Hutter, 2017), annealing towards 10^{-8}.

We apply a variety of common data augmentation techniques which differs from previous NAS benchmarks where the training accuracy of many evaluated architectures reached 100% (Ying et al., 2019; Dong & Yang, 2020) indicating overfitting on the training set. We used CutOut (DeVries & Taylor, 2017) with cutout length 16 and MixUp (Zhang et al., 2018) with alpha 0.2. For regularization, we used an auxiliary tower (Szegedy et al., 2015) with a weight of 0.4 and DropPath (Larsson et al., 2017) with drop probability of 0.2. We trained each architecture for 100 epochs with a batch size of 96, using 32 initial channels and 8 cell layers. We chose these values to be close to the proxy model used by DARTS while also achieving good performance.

D Surrogate Model Analysis

D.1 Preprocessing of the graph topology

DGN preprocessing All DGN were implemented using PyTorch Geometric (Fey & Lenssen, 2019) which supports the aggregation of edge attributes. Hence, we can naturally represent the DARTS architecture cells, by assigning the embedded operations to the edges. The nodes are labeled as input, intermediate and output nodes. We represent the DARTS graph as shown in Figure 14, by connecting the output node of each cell type with the inputs of the other cell, allowing information from both cells to be aggregated during message passing. Note the self-loop on the output node of the normal cell, which we found necessary to get the best performance.

Preprocessing for other surrogate models Since we make use of the framework implemented by BOHB (Falkner et al., 2018) to easily parallelize the architecture search algorithms across many compute nodes, we also represent our search space using ConfigSpace (Lindauer et al., 2019). For all non-DGN based surrogate models, we use the vector representation of a configuration given by ConfigSpace as input to the model.

D.2 Details on the GIN

The GIN implementation on the Open Graph Benchmark (OGB) (Hu et al., 2020) uses virtual nodes (additional nodes which are connected to all nodes in the graph) to boost performance as well as generalization and consistently achieves good performance on their public leaderboards. Other GNNs from Errica et al. (2020), such as DGCNN and DiffPool, performed worse in our initial experiments and are therefore not considered.

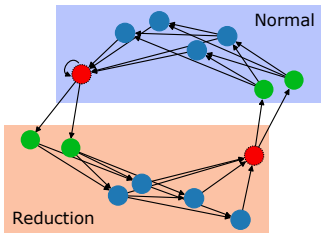


Figure 14: Architecture with inputs in green, intermediate nodes in blue and outputs in red.

Following recent work in Predictor-based NAS (Ning et al., 2020; Xu et al., 2019b), we use a per batch ranking loss because the ranking of an architecture is equally important to an accurate prediction of the validation accuracy in a NAS setting. We use the ranking

[6] We chose DE because it both explored and exploited well, see Figure 8 in the Appendix.

loss formulation by GATES (Ning et al., 2020) which is a hinge pair-wise ranking loss with margin m=0.1.

D.3 Details on HPO for surrogate models

All hyperparameters of the surrogate models were tuned using BOHB (Falkner et al., 2018) as a black-box optimizer; details on their respective hyperparameter search spaces are given in Table 5. We use train/val/test splits (0.8/0.1/0.1) stratified by the NAS methods used for the data collection.

D.4 HPO for runtime prediction model

Our runtime prediction model is an LGB model trained on the runtimes of architecture evaluations of DE. This is because we partially evaluated the architectures utilizing different CPUs. Hence, we only choose to train on the evaluations carried out by the same optimizer on the same hardware to keep a consistent estimate of the runtime. DE is a good choice in this case because it both explored and exploited the architecture space well. The HPO space used for the LGB runtime model is the same used for the LGB surrogate model.

D.5 Leave One-Optimizer-Out Analysis

Since the aim of NAS-Bench-301 is to allow efficient benchmarking of novel NAS algorithms, it is necessary to ensure that the surrogate model can deliver accurate performance estimation on data from trajectories by unseen NAS methods. Similarly to Eggensperger et al. (2015), we therefore perform a form of cross-validation on the optimizers we used for data collection, i.e. we leave out all data collected by one of the NAS methods entirely during training (using a stratified 0.9/0.1 train/val split over the other NAS methods). Then, we predict the unseen results from the left-out NAS method to evaluate how well the models extrapolate to the region covered by the 'unseen' method. We refer to this as the leave-one-optimizer-out (LOOO) setting.

Results The results in Table 6 show that the rank correlation between the predicted and observed validation accuracy remains high even when a well-performing optimizer such as RE is left out. Predicting BANANAS in the LOOO fashion yields a lower rank correlation, because it focuses on well-performing architectures that are harder to rank; however, the high R^2 shows that the fit is still good.

Conversely, leaving out DARTS causes a low R^2 but still high sKT; this is due to architectures with many skip connections in the DARTS data that are overpredicted (further discussed in Section E.1). For full details, Figure 15 in the appendix provides scatter plots of the predicted vs. true performance for each NAS method. A detailed scatter plot of the predicted performance against the true performance for each optimizer and surrogate model in an LOOO analysis is provided in Figure 15.

D.6 Parameter-free Operations

Several works have found that methods based on DARTS (Liu et al., 2019) are prone to finding sub-optimal architectures that contain many, or even only, parameter-free operations (max. pooling, avg. pooling or skip connections) and perform poorly (Zela et al., 2020a). We therefore evaluated the surrogate models on such architectures by replacing a random selection of operations in a cell with one type of parameter-free operations to match a certain ratio of parameter-free operations in a cell. This analysis is carried out over the test set of the surrogate models and hence contains architectures collected by all optimizers. For a more robust analysis, we repeated this experiment 4 times for each ratio of operations to replace.

Results Figure 16 shows that both the GIN and the XGB model correctly predict that the accuracy drops with too many parameter-free operations, particularly for skip connections. The groundtruth of architectures with only parameter-free operations is displayed as scatter plot. Out of the two models, XGB captures the slight performance improvement of using a few skip connections better. LGB failed to capture this trend but performed very similarly to XGB for the high number of parameter-free operations.

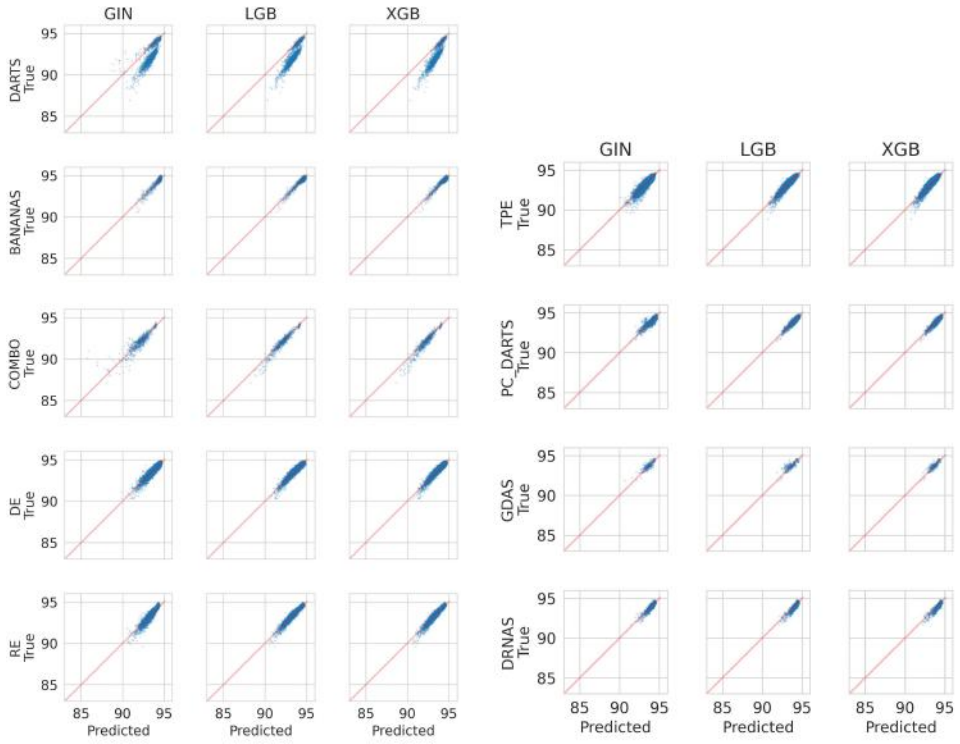
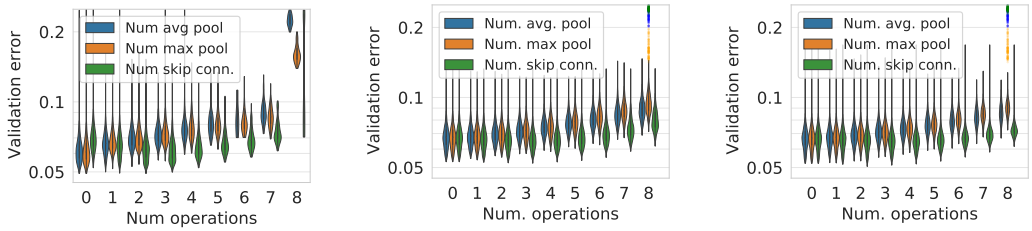


Figure 15: Scatter plots of the predicted performance against the true performance of different surrogate models on the test set in a Leave-One-Optimizer-Out setting.



(a) NAS-Bench-301 dataset (b) GIN (c) XGB

Figure 16: (Left) Distribution of validation error in dependence of the number of parameter-free operations in the normal cell on the NAS-Bench-301 dataset. (Middle and Right) Predictions of the GIN and XGB surrogate model. The collected groundtruth data is shown as scatter plot. Violin plots are cut off at the respective observed minimum and maximum value.

E Benchmark Analysis

E.1 One-Shot Trajectories

NAS-Bench-301 can also be used to monitor the behavior of one-shot NAS optimizers throughout their search phase, by querying the surrogate model with the currently most promising discrete architecture. This can be extremely useful in many scenarios since uncorrelated proxy and true objectives can lead to potential failure modes, e.g., to a case where the found architectures contain only skip connections in the normal cell (Zela et al., 2020a,b; Dong & Yang, 2020) (we study such a failure case in Appendix E.2 to ensure robustness of the surrogates in said case). We demonstrate this use case in a similar LOOO analysis as for the black-box optimizers, using evaluations of the discrete architectures from each search epoch of multiple runs of DARTS, PC-DARTS and GDAS

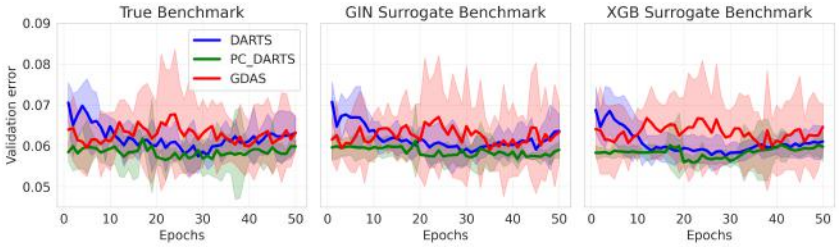


Figure 17: Anytime performance of one-shot optimizers, comparing performance achieved on the real benchmark and on surrogate benchmarks built with GIN and XGB in a LOOO fashion.

as ground-truth. We performed 5 runs for each optimizer with 50 search epochs and evaluated the architecture obtained by discretizing the one-shot model at each search epoch. Figure 17 shows that the surrogate trajectories closely resemble the true trajectories.

E.2 Diverging One-Shot Methods

For DARTS, in addition to the default search space, we collected trajectories on the constrained search spaces from Zela et al. (2020a) to cover a failure case where DARTS diverges and finds architectures that only contain skip connections in the normal cell. To show that our benchmark is able to predict this divergent behavior, we show surrogate trajectories when training on all data, when leaving out the trajectories under consideration from the training data, and when leaving out all DARTS data in Figure 18.

While the surrogates model the divergence in all cases, they still overpredict the architectures with only skip connections in the normal cell especially when leaving out all data from DARTS. The bad performance of these architectures is predicted more accurately when including data from other DARTS runs. This can be attributed to the fact that the surrogate models have not seen any, respectively very few data, in this region of the search space. Nevertheless, it is modeled as a bad-performing region and we expect that this could be further improved on by including additional training data accordingly, since including all data in training shows that the models are capable to of capturing this behavior.

F Guidelines on using NAS-Bench-301

We want to mention the risk that prior knowledge about the surrogate model in NAS-Bench-301 could lead to the design of algorithms that may overfit to the surrogate benchmark. To this end, we recommend the following best practices to ensure a safe and fair benchmarking of NAS methods on NAS-Bench-301 and future surrogate benchmarks:

- The surrogate model should be treated as a black-box function, hence only be used for performance prediction and not exploited to extract, e.g., gradient information.

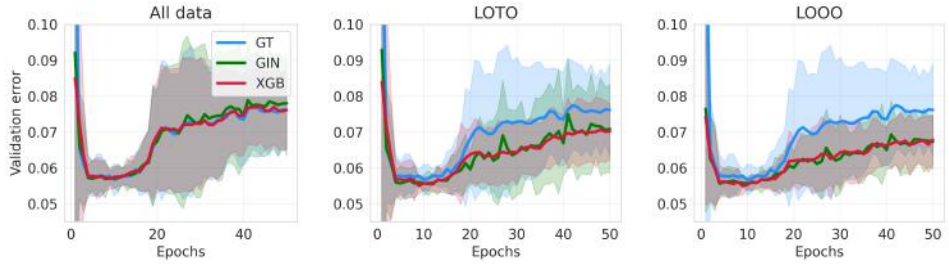


Figure 18: Groundtruth (GT) and surrogate trajectories on a constrained search space where the surrogates are trained with all data, leaving out the trajectories under consideration (LOTO), and leaving out all DARTS architectures (LOOO).

- We discourage benchmarking methods that internally use the same model as the surrogate model picked in NAS-Bench-301 (e.g. GNN-based Bayesian optimization should not only be benchmarked using the GIN surrogate benchmark).
- In order to ensure comparability of results in different published papers, we ask users to state the benchmark's version number. We will continuously collect more training data and further improve the surrogate model predictions. So far, we release NB301-XGB-v1.0 and NB301-GIN-v1.0.

G Guidelines for Creating Surrogate Benchmarks

In order to help with the design of realistic surrogate benchmarks in the future, we provide the following list of guidelines:

- Data Collection: The data collected for the NAS benchmark should provide (1) a good overall coverage, (2) explore strong regions of the space well, and (3) optimally also cover special areas in which poor generalization performance may otherwise be expected.
    1. A good overall coverage can be obtained by random search (as in our case), but one could also imagine using better space-filling designs or adaptive methods for covering the space even better. In order to add additional varied architectures, one could also think about fitting one or more surrogate models to the data collected thus far, finding the regions of maximal predicted uncertainty, evaluate architectures there and add them to the collected data, and iterate. This would constitute an active learning approach.
    2. A convenient and efficient way to identify regions of strong architectures is to run NAS methods. In this case, the found regions should not only be based on the strong architectures one NAS method finds but rather on a set of strong and varied NAS methods (such as, in our case, one-shot methods and different types of discrete methods, such as Bayesian optimization and evolution). In order to add additional strong architectures, one could also think about fitting one or more several surrogate models to the data collected thus far, finding the predicted optima of these models, evaluate and add them to the collected data and iterate. This would constitute a special type of Bayesian optimization.
    3. Special areas in which poor generalization performance may otherwise be expected may, as in our case, e.g., include architectures with many parameterless connections, and in particular, skip connections. Other types of failure modes the community learns about would also be useful to cover.
- Surrogate Models: As mentioned in the guidelines for using a surrogate benchmark (see Section 7), benchmarking an algorithm that internally uses the same model type as the surrogate model should be avoided. Therefore, to provide a benchmark for a diverse set of algorithms, we recommend providing different types of surrogate models with a surrogate benchmark.
- Verification: As a means to verify surrogate models, we stress the importance of leave-one-optimizer-out experiments both for data fit and benchmarking, which simulate the benchmarking of 'unseen' optimizers.
- Since most surrogate benchmarks will continue to grow for some time after their first release, to allow apples-to-apples comparisons, we strongly encourage to only release surrogate benchmarks with a version number.
- In order to allow the evaluation of multi-objective NAS methods, we encourage the logging of as many relevant metrics of the evaluated architectures other than accuracy as possible, including training time, number of parameters, and multiply-adds.
- Alongside a released surrogate benchmark, we strongly encourage to release the training data its surrogate(s) were constructed on, as well as the test data used to validate it.
- In order to facilitate checking hypotheses gained using the surrogate benchmarks in real experiments, the complete source code for training the architectures should be open-sourced alongside the repository, allowing to easily go back and forth between querying the model and gathering new data.

Model	Hyperparameter	Range	Log-transform	Default Value
GIN	Hidden dim.	[16, 256]	true	24
	Num. Layers	[2, 10]	false	8
	Dropout Prob.	[0, 1]	false	0.035
	Learning rate	[1e-3, 1e-2]	true	0.0777
	Learning rate min.	const.	-	0.0
	Batch size	const.	-	51
	Undirected graph	[true, false]	-	false
	Pairwise ranking loss	[true, false]	-	true
	Self-Loops	[true, false]	-	false
	Loss log transform	[true, false]	-	true
	Node degree one-hot	const.	-	true
BANANAS	Num. Layers	[1, 10]	true	17
	Layer width	[16, 256]	true	31
	Dropout Prob.	const.	-	0.0
	Learning rate	[1e-3, 1e-1]	true	0.0021
	Learning rate min.	const.	-	0.0
	Batch size	[16, 128]	-	122
	Loss log transform	[true, false]	-	true
	Pairwise ranking loss	[true, false]	-	false
XGBoost	Early Stopping Rounds	const.	-	100
	Booster	const.	-	gbtree
	Max. depth	[1, 15]	false	13
	Min. child weight	[1, 100]	true	39
	Col. sample bylevel	[0.0, 1.0]	false	0.6909
	Col. sample bytree	[0.0, 1.0]	false	0.2545
	lambda	[0.001, 1000]	true	31.3933
	alpha	[0.001, 1000]	true	0.2417
	Learning rate	[0.001, 0.1]	true	0.00824
LGBoost	Early stop. rounds	const.	-	100
	Max. depth	[1, 25]	false	18
	Num. leaves	[10, 100]	false	40
	Max. bin	[100, 400]	false	336
	Feature Fraction	[0.1, 1.0]	false	0.1532
	Min. child weight	[0.001, 10]	true	0.5822
	Lambda L1	[0.001, 1000]	true	0.0115
	Lambda L2	[0.001, 1000]	true	134.5075
	Boosting type	const.	-	gbdt
	Learning rate	[0.001, 0.1]	true	0.0218
Random Forest	Num. estimators	[16, 128]	true	116
	Min. samples split.	[2, 20]	false	2
	Min. samples leaf	[1, 20]	false	2
	Max. features	[0.1, 1.0]	false	0.1706
	Bootstrap	[true, false]	-	false
ϵ-SVR	C	[1.0, 20.0]	true	3.066
	coef. 0	[-0.5, 0.5]	false	0.1627
	degree	[1, 128]	true	1
	epsilon	[0.01, 0.99]	true	0.0251
	gamma	[scale, auto]	-	auto
	kernel	[linear, rbf, poly, sigmoid]	-	sigmoid
	shrinking	[true, false]	-	false
	tol	[0.0001, 0.01]	-	0.0021
μ-SVR	C	[1.0, 20.0]	true	5.3131
	coef. 0	[-0.5, 0.5]	false	-0.3316
	degree	[1, 128]	true	128
	gamma	[scale, auto]	-	scale
	kernel	[linear, rbf, poly, sigmoid]	-	rbf
	nu	[0.01, 1.0]	false	0.1839
	shrinking	[true, false]	-	true
	tol	[0.0001, 0.01]	-	0.003

Table 5: Hyperparameters of the surrogate models and the default values found via HPO.

	Model	No RE	No DE	No COMBO	No TPE	No BANANAS	No DARTS	No PC-DARTS	No DrNAS	No GDAS
R^2	LGB	**0.917**	**0.892**	**0.919**	**0.857**	0.909	-0.093	**0.826**	0.699	0.429
	XGB	0.907	0.888	0.876	0.842	**0.911**	-0.151	0.817	0.631	**0.672**
	GIN	0.856	0.864	0.775	0.789	0.881	**0.115**	0.661	**0.790**	0.572
sKT	LGB	**0.834**	**0.782**	**0.833**	**0.770**	0.592	**0.780**	**0.721**	0.694	0.595
	XGB	0.831	0.780	0.817	0.762	**0.596**	0.775	0.710	**0.709**	**0.638**
	GIN	0.798	0.757	0.737	0.718	0.567	0.765	0.645	0.706	0.607

Table 6: Leave One-Optimizer-Out performance of the best surrogate models.