A Design space

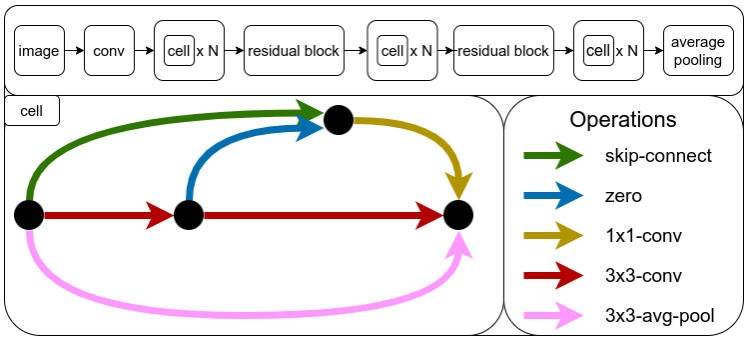

Figure 6: An illustration of NAS-Bench-201 cell-based architecture search space, adapted from Dong and Yang [6].

B Performance dataset

Documentation and code For documentation on accessing and using the performance dataset see `https://automl.github.io/jahs_bench_201/performance_dataset/` and for the corresponding code see `https://github.com/automl/jahs_bench_201`. Our hosting plan for the relevant assets can be found at `https://automl.github.io/jahs_bench_201/performance_dataset/`.

License: We release the code used to build our benchmark and perform our experiments under the MIT License (`https://mit-license.org/`), whereas we release data we created, including the performance metrics collected by us, the splits used to train, validate and test our surrogate models, and our surrogate models, under the CC BY 4.0 License (`https://creativecommons.org/licenses/by/4.0/`).

Deep learning pipeline We largely follow the same pre-processing procedure as was used by Dong and Yang [6] for training every sampled configuration in our performance dataset. For CIFAR-10, we use the exact same training/validation/testing splits as Dong and Yang [6] in order to establish fair comparison, but opted for different splits for Colorectal-Histology and Fashion-MNIST in order to introduce variety in our tasks. The data augmentation procedures are the same for all three datasets and mirror those used by Dong and Yang [6]. When Trivial Augment is enabled in a configuration, it is applied before applying the other augmentation operations. Following Dong and Yang [6], we used stochastic gradient descent with Nesterov momentum [56] using a fixed momentum of 0.9 and a fixed minibatch size of 256 for all of our configurations. We used Cosine Annealing[57] without restarts with a maximum epochs parameter value of 200 for all configurations. The sampled learning rate was used as the initial learning rate for the scheduler.

Compute resources We trained the configurations on a large SLURM-based cluster with approximately 300,000 CPU-cores available in parallel. Each configuration was trained on a single CPU-core of an Intel Skylake Xeon Platinum 8174 processor. Our total compute consumption was approximately 7×10^6 core-hours.

Sampling In order to maintain statistical independence of our samples across multiple parallel workers and repeatability of our experiments, a Philox counter-based pseudo-random number generator [58] was used to sample the configurations as well as a unique random seed per configuration to be used for initializing the neural network. We note, however, that due to the complex interactions of programming environments and hardware, it is not possible to guarantee that the initial states of the models can always be reproduced, even when these seeds are available.

C Surrogate creation

Metrics Since a subset of the metrics present in the performance dataset can be subsumed by others, we chose to restrict the metrics that the surrogate can predict to the top-1 accuracy of the training, validation and test splits, the size of the model in memory, the average latency of the model computed across the validation and test splits, the number of floating point operations per second (FLOPS) performed by a forward pass of the model, and the total runtime, for a total of 7 metrics. We trained a separate model for each objective.

Data splits We split our performance dataset into a training, validation and test split in an approximately 70-15-15 ratio. We attempted to maintain the sampling distribution in terms of the number of configurations per fidelity group between the three data splits. This was achieved by a two-step approach towards split generation. In step 1, we treated every single configuration's data points across multiple epochs as time-series data, where each epoch is a single time step, thereby grouping together all the data from every epoch that a given configuration was evaluated for, treating them as a single data point for the purpose of split-generation. In the second step, we enforced the resultant sampling distribution of grouped data points across fidelity groups (recall, we sampled 10,000 configurations per fidelity group) when generating the training, validation and testing splits. This ensures that all three data splits retain all or most of the statistical properties, including any biases, of the original performance dataset.

Adding bounds Since XGBoost is an unbounded regression model i.e. its codomain is $\mathbb{R}$, it was necessary to impose relevant domain bounds on the predicted metrics. For this purpose, during training we first fit and applied a parameterised linear scaling transformation $f_s : \mathbb{R}_Y \to [\delta, 1-\delta]$ to the observed metric values Y and obtained scaled values $Y_s = f_s(Y)$, where $\delta \to 0$ is a very small value used for numerical stability and $\mathbb{R}_Y$ is the range of values observed in the training set. Following this, we applied an inverse sigmoid transformation $\sigma^{-1} : (0,1) \to \mathbb{R}$ to obtain the final training labels $Y_\sigma = \sigma^{-1}(Y_s) = (\sigma^{-1} \circ f_s)(Y)$. At the time of prediction using the surrogate, we applied the inverse of these transformations in the opposite sequence so as to obtain the required predicted performance metric values, i.e. the model prediction $\hat{Y} = (f_s^{-1} \circ \sigma)(\tilde{Y})$, where $\tilde{Y}$ are the raw predictions generated by XGBoost, $\sigma : \mathbb{R} \to (0,1)$ is the sigmoid function and $f_s^{-1} : [\delta, 1-\delta] \to \mathbb{R}_Y$ is the inverse of the scaling function f_s.

Hyperparameter optimization We optimize over the same hyperparameter space as Mehta et al. [24]. Whereas fitting XGBoost used mean-squared-error as a regression metric, quality of fit for hyperparameters was judged using Kendall's tau rank correlation values. We use Bayesian optimization (BO) [26] with user priors [27] with the NePS package [28]. For each surrogate model, 20 configurations were evaluated using 4 workers in parallel.

Compute resources We trained our surrogates using a SLURM-based cluster of NVidia GeForce RTX2080 GPUs, consuming approximately 200 GPU-hours of compute in the process, including the training of all configurations queried by the HPO process.

Documentation and code For documentation on creating surrogates see `https://automl.github.io/jahs_bench_201/surrogate/`. For using our benchmark this is not required, but we provide this to improve reproducibility. For the corresponding code see `https://github.com/automl/jahs_bench_201`.

D Experiment details

To facilitate a uniform comparison of different optimizers using JAHS-Bench-201, we propose to study incumbent trajectories of validation loss against the runtime equivalent to 100 evaluations in the maximum fidelity setting. This allows for a comprehensive analysis of optimizer's performance in a practical setting of bounded compute resources. Furthermore, we propose to scale the runtime by an average configuration evaluation runtime (see Table 8) to simplify the comparison across all tasks and help interpretability. We provide both experimental and analysis scripts in our repository (`https://github.com/automl/jahs_bench_201_experiments`). We estimate the cumulative

Table 8: The average predicted runtime of the surrogate models of a full function evaluation across 10^4 random configurations for each of the datasets.

Dataset	Average predicted runtime [CPU-d]
CIFAR-10	2.0
Colorectal-Histology	0.2
Fashion-MNIST	2.2

Table 9: We report speedups of model-free JAHS for reaching the same final performance as its competitors.

Task	Speedup over HPO-only	Speedup over NAS-only
CIFAR-10	$54.7\times$	$33.7\times$
Colorectal-Histology	$75.2\times$	$20.1\times$
Fashion-MNIST	$8.5\times$	$34.6\times$
Geometric mean	$32.7\times$	$28.6\times$

resource consumption for our experiments performed on Intel(R) Xeon(R) Gold 6242 CPU @ 2.80GHz to be 1.75 CPU-core-hours.

E Broader Impact And CO_2 Emissions

Carbon Footprint. Even though creating our neural network performance dataset required a substantial amount of compute, approximately 7×10^6 core-hours, the computations remain carbon neutral since the compute cluster used by us uses 100% clean energy. This does not take into account carbon emissions for optimizing and training the surrogate benchmarks based on the data and indirect emissions such as creating the compute hardware and maintenance of the compute cluster. Further, as using our surrogate benchmarks is much less compute intensive than the corresponding trainings, our work contributes to a reduction in carbon emissions for JAHS research.

Democratization of JAHS With the help of JAHS-Bench-201, researchers no longer need access to expensive, high performance compute clusters and can instead perform JAHS research using commercial laptops. This democratization of JAHS could lead to rapid advancements and increase the utility and pervasiveness of Deep learning in general. As advancements in JAHS research are of a foundational nature, therefore they can lead to both good and bad societal impacts.

No human subjects and personal data Our work did not involve any aspects of human experimentation or handling of personal data, thereby making any direct or indirect negative consequences pertaining to issues of privacy, personal health, safety, discrimination and other related aspects arising from our work very unlikely.

F Colorectal-Histology Data Distribution Shift

We noted that our surrogate models' performance on the Colorectal-Histology task was much worse than on CIFAR-10 and Fashion-MNIST. In order to investigate the cause behind this, we performed two additional experiments.

In the first experiment, 20 configurations were randomly chosen from the set of configurations belonging to the highest fidelity group (N=5, W=16, R=1.0) that had already been evaluated on CIFAR-10 and Colorectal-Histology for our performance dataset and re-evaluated for 200 epochs using 2 different, randomly sampled sets of seeds for initialization. Then, we calculated the rank correlation of the configurations for each of the metrics our surrogates were trained to predict across different initial conditions for the same configuration at the same epoch of evaluation. This enabled us to coarsely estimate the amount of noise in the underlying metric data distribution to be expected based on the initial conditions. We present the results of this analysis in Table 11.

Table 10: We report speedups of model-based JAHS for reaching the same final performance as its competitors.

Task	Speedup over HPO-only	Speedup over NAS-only
CIFAR-10	$48.6\times$	$33.7\times$
Colorectal-Histology	$64.1\times$	$15.7\times$
Fashion-MNIST	$2.5\times$	$20.2\times$
Geometric mean	$19.8\times$	$22.0\times$

The task-agnostic metrics FLOPS and size exhibited perfect rank-correlation whereas the total runtime and latency metrics exhibited similar levels of noise in the underlying data distribution, as expected. However, the task-dependent accuracy metrics exhibit clear differences between the two tasks, wherein we observe far lower rank correlation between different initializations for models evaluated on Colorectal-Histology than on CIFAR-10. This leads us to believe that the underlying data-distribution for these metrics is much noisier in the case of evaluations on Colorectal-Histology. This presents a strong case for the reason behind the difference in performance of some of our surrogate models, but it is insufficient to explain why this is also the case for the other half of the surrogates.

Table 11: Kendall's τ correlation values for the metric values recorded between two different initializations of the same model configurations that were then evaluated on the tasks CIFAR-10 and Colorectal-Histology.

	CIFAR-10	Colorectal-Histology
FLOPS	1.000	1.000
Latency	0.523	0.547
Runtime	0.843	0.847
Size	1.000	1.000
Train Acc	0.966	0.886
Valid Acc	0.866	0.718
Test Acc	0.865	0.720

In the second experiment, we re-ran all the configurations evaluated on Colorectal-Histology for exactly 1 epoch under different initial conditions and re-recorded the value of the FLOPS metric for these configurations. We observed an overall Kendall's-τ value of merely 0.695729, indicating an issue with the recorded metric values. After repeated tests of our training pipeline, we attribute this shift in the data distribution to a change in the underlying hardware or firmware environment of our compute cluster that we were unaware of over the course of the Colorectal-Histology performance data collection, which is the only task for which data was collected across two batches with a separation of several months instead of one. We present a further breakdown of how this affects our data in Table 12. We note that we have no cause to suspect any such issues in the CIFAR-10 and Fashion-MNIST performance datasets. We also note that this shift is only expected to affect those metrics which are hardware-dependent, i.e. the FLOPS along with the "Runtime" and "Duration" metrics.

Table 12: Kendall's τ correlation values between the FLOPS metric values recorded in the existing performance dataset for models evaluated on the Colorectal-Histology task and the newly generated FLOPS values, within each fidelity group.

N	Resolution W	0.25	0.50	1.00
1	4	0.488	0.744	0.647
	8	0.624	0.992	0.991
	16	0.928	0.983	0.970
3	4	0.993	0.989	0.988
	8	0.792	0.724	0.733
	16	0.983	0.900	0.993
5	4	0.897	0.987	0.902
	8	0.900	0.980	0.657
	16	0.422	0.961	0.996