In-Context Learning for Latency Estimation

Timur M. Carstensen[1*] **Thomas Elsken**[3†] **Martin Rapp**[2]

[1]University of Freiburg, Germany
[2]Bosch Center for Artificial Intelligence (BCAI), Renningen, Germany
[3]Recogni Inc., Munich, Germany

Abstract Neural architectures must be computationally efficient for edge device deployment to perform well under hardware constraints. Current Hardware-Aware NAS (HW-NAS) methods use surrogate models to predict hardware metrics (e.g., latency) during architecture search. These surrogate models typically require large amounts of data to train or finetune and rely on search-space specific encodings and meta-learning. We propose an In-Context Learning-based method for hardware latency estimation that generalises to unseen hardware in a single forward pass with a few labelled samples. Our surrogate is trained on real architecture-latency pairs with data augmentation to improve sample efficiency. Our method surpasses the state-of-the-art in Spearman's ρ and is up to 72% to 92% more sample efficient.

1 Introduction

Hardware-aware neural architecture search (HW-NAS) (Benmeziane et al., 2021) tackles the problem of automatically designing neural architectures that both achieve a high task performance (e.g., high accuracy) and run efficiently on the hardware (e.g., low latency). HW-NAS is typically formulated as an iterative optimization procedure (e.g., using evolutionary search) that may evaluate more than $10,000$ neural architectures (Real et al., 2017; Pham et al., 2018). Profiling such a large number of neural architectures on real hardware to obtain latency measurements represents a major bottleneck to HW-NAS (Benmeziane et al., 2021; Cai et al., 2018). This is because it may take minutes to compile and deploy a single neural architecture, and it is difficult to parallelize due to limited number of physical hardware boards and license restrictions in the required deployment tools.

Surrogate models allow to predict the latency of a certain neural architecture instead of profiling it on real hardware. For instance, Cai et al. (2019) fit a small neural network to predict the latency given a vectorized representation of the neural architecture. Such models are specific to a certain hardware target and NAS search space. Zhang et al. (2021) fit detailed models that predict the latency of individual neural building blocks such as layers. This allows to predict the latency of a large variety of neural architectures, but still is specific to a single hardware target. Both works require substantial amounts of training data that needs to be obtained first whenever a new hardware target is observed. Lee et al. (2021) and Akhauri and Abdelfattah (2023) aim at generalizing across different hardware targets by formulating the problem as a few-shot regression problem with meta-learning, where the prediction model is finetuned on few data samples for the specific hardware target at hand. However, still substantial amounts of training data are required for pre-training the regression model.

We introduce a novel hardware surrogate model to predict the latency of new neural architectures on new hardware targets that requires a) less training data for training and b) fewer

*This work was done as part of a master's thesis at the Bosch Center for Artificial Intelligence (BCAI).
†Work done while at the Bosch Center for Artificial Intelligence (BCAI).

latency samples on a new hardware target. This is achieved by 1) formulating the problem as an in-context learning problem and 2) pre-training the predictor model on synthetic hardware targets that augment the real measurement data.

2 Method

2.1 Problem formulation

We aim at predicting the latency of a given neural architecture $A \in \mathcal{A}$ (query) on a hardware device $h \in \mathcal{H}$. Additionally, a set of n support examples $\mathcal{D}$ is provided, where each example comprises a neural architecture and its latency on hardware h: $\mathcal{D} = \{(A_i, y_i) : i \in \{1, \ldots, n\}\}$. We formulate this problem as an in-context learning problem, and fit a regression model $f(A, \mathcal{D}; \theta)$, parameterized by θ, once, and do not require any fine-tuning to a specific hardware afterwards.

2.2 Neural architecture encoding

Following Akhauri and Abdelfattah (2023), we use Zero-Cost Proxies (ZCPs) to encode neural architectures. Since ZCPs have been introduced as predictors of model performance in terms of final validation accuracy, it is not directly obvious why they should inform us about an architecture's hardware-latency. Empirically, we can show that some proxies such as l2_norm, FLOPs, # params, and synflow (Tanaka et al., 2020) have consistently high rank correlation with hardware-latencies across devices (see Appendix B). Contrary to the final model presented in Akhauri and Abdelfattah (2023) and Lee et al. (2021), we do not additionally use search-space specific representations as an architecture encoding (White et al., 2023). The main benefit of this approach is that the input dimension to our model stays the same when moving to a new search space, given that one uses the same ZCPs. The ZCPs for the search spaces we consider are a subset of those covered in Krishnakumar et al. (2022). See Appendix C for a complete list.

More concretely, we represent each architecture A in a search-space $\mathcal{A}$ as a vector of ZCP measurements:

$$\boldsymbol{x}_A = \{zcp_1(A), zcp_2(A), \ldots, zcp_k(A)\} \tag{1}$$

2.3 Surrogate model architecture

The architecture of the surrogate model is a Transformer with masked attention that only allows attention between the query architecture A to support examples in $\mathcal{D}$, similar to the architecture used by Prior-data Fitted Networks (PFN) (Müller et al., 2022). Finally, we formulate latency prediction as a classification problem, where we divide the latency range for the current hardware into b buckets whose borders are chosen such that they contain an equal number of samples under the prior data.

2.4 Synthetic data priors

The priors are used to generate synthetic samples to train our model more effectively with limited amounts of latency measurements on real hardware (HW). Here, a sample refers to a set of architecture-latency pairs (datapoints) on some device j which is divided into a support and query set of size n and m, respectively. To construct a batch, we repeatedly sample from the respective prior until we reach our desired batch size.

The priors employed in our method use *real* latency data and can thus be seen as a form of data augmentation (Jaitly and Hinton, 2013), deviating from the fully synthetic priors in the original PFN (Müller et al., 2022). Our priors differ in the choice of the augmentation method that acts on the sampled architectures A, corresponding latencies $\mathcal{L}$ across the set of devices D and ZCPs:

1. **Latency scaling**: this prior scales the latency values by a random factor $x \sim \mathcal{U}(0, 1)$.

2. **Synthetic HW**: this prior creates a synthetic hardware target. Latency values are computed by a weighted average between several random hardware targets.

3. **Synthetic HW + ZCPs**: this prior extends the synthetic hardware prior by a random linear combination of ZCPs. This is motivated by the fact that some ZCPs are cheap to obtain and have been shown to correlate with hardware-latency and could thus be good predictors thereof. This prior is shown in Algorithm 1.

Algorithm 1 Linear combination in latencies and ZCPs (Synthetic HW + ZCP)

Require: architecture representation $\mathbf{X}$, function $L(a, d)$, function $Z(a)$, set of devices $\mathcal{H}$, set of architectures A

sample $\alpha \sim \mathcal{U}(0, 1)$
sample $H \in \mathcal{H}$ with $|H| \sim \mathcal{U}(1, |\mathcal{H}|)$
sample $K \in ZCP$ with $|K| \sim \mathcal{U}(1, \#ZCPs)$
sample $\mathbf{w} \sim \mathcal{U}(0, 1)^{|H|+|K|}$
$\mathbf{w} \leftarrow \text{Softmax}(\mathbf{w})$ ▷ ensure that $\sum_{i=1}^{|H|} w_i = 1$
$\hat{y} \leftarrow []$
for each architecture a in A **do**
    $y = \alpha * \sum_{i=1}^{|H|} w_i * L(a, H_i) + \sum_{j=|H|+1}^{|H|+|K|} w_j * Z(a)_{K_j}$
    append y to $\hat{y}$
end for
return $\{\mathbf{X}, \hat{\mathbf{y}}\}$

3 Experiments

To evaluate the efficacy of our method, we use the same latency datatsets as *HELP* (Lee et al., 2021): 421, 875/135, 000 latency measurements across 27 hardware devices from four classes (GPU, CPU, mobile, and embedded) for the NAS-Bench-201 and FBNet search spaces (see Appendix A for a complete list of hardware devices). We also follow the same data splits as *HELP*, using 900 and 4, 000 architectures in our training set for NB201 and FBNet, respectively, and 14, 725 and 1, 000 for the validation and test sets. While we had access to latency measurements for all 15, 625 architectures for NAS-Bench-201, we only had measurements for 5, 000 out of all 10^{21} architectures in the FBNet search space. We use the same hardware device split as *HELP* (see Appendix A). To make our results comparable to that of *HELP*, we use 20 support and 1 query datapoint to match the number of latency samples used for hardware encoding (10) and few shot adaptation (10). We evaluate the performance of our method using Spearman's ρ which quantifies the rank correlation between predicted and ground truth latencies. Rank correlation is particularly relevant when deploying this method in hardware-aware NAS, where the goal is to find the optimal architecture. For each reported result we report the mean and standard deviation across 5 random seeds.

3.1 Latency estimation

We trained the model for 50, 000 iterations and evaluated it on the test set for each prior. The results for NB201 can be seen in Table 1. Our model trained on the Synthetic HW + ZCP was able to outperform all previous relevant works on HW-latency surrogate models in terms of Spearman's ρ for the test device and architecture set for the NB201 search space when taking the mean across test devices. In particular we achieved a 2.2% improvement in mean Spearman's ρ over *HELP*, going from 0.932 to 0.953. The greatest per-device performance uplift was achieved for the Pixel2, going from a Spearman's ρ of 0.802 to 0.898 (model trained with scaling-prior), which constitutes a 11.97% increase in performance. On a per-device level, we are able to outperform *HELP* on all

Table 1: Comparison of the latency estimators on unseen devices for NAS-Bench-201 in terms of Spearman's ρ. Values for FLOPS, Layer-wise Predictor, BRP-NAS, and HELP obtained from Table 3 in Lee et al. (2021). Mean ± (standard deviation) over 5 random seeds

Method	Transfer	Sample	Unseen Device			Mean
			GPU	CPU	Pixel2	
FLOPS	-	-	0.950	0.826	0.765	
Layer-wise Predictor	-	-	0.667	0.866	-	
BRP-NAS (Dudziak et al., 2020)	-	900	0.814	0.796	0.666	
BRP-NAS(+extra samples)	-	3200	0.822	0.805	0.693	
HELP (Lee et al., 2021)	✓	20	0.987	**0.989**	0.802	
PFN (no prior)	✓	20	0.909 ± (0.01)	0.931 ± (0.009)	0.828 ± (0.02)	
PFN (latency scaling)	✓	20	0.978 ± (0.005)	0.979 ± (0.003)	**0.898** ± (0.002)	
PFN (Synthetic HW)	✓	20	0.985 ± (0.002)	0.983 ± (0.001)	0.895 ± (0.002)	
PFN (Synthetic HW + ZCP)	✓	20	**0.991** ± (0.002)	0.983 ± (0.001)	0.885 ± (0.003)	

Method	Unseen Device			Mean
	Raspi4	Eyeriss	FPGA	
FLOPS	0.846	0.437	0.900	0.787
Layer-wise Predictor	-	-	-	0.767
BRP-NAS (Dudziak et al., 2020)	0.847	0.811	0.801	0.789
BRP-NAS(+extra samples)	0.853	0.830	0.828	0.805
HELP (Lee et al., 2021)	0.890	0.940	0.985	0.932
PFN (no prior)	0.867 ± (0.01)	0.655 ± (0.04)	0.925 ± (0.01)	0.8529 ± (0.009)
PFN (latency scaling)	0.918 ± (0.006)	0.721 ± (0.07)	0.982 ± (0.003)	0.912 ± (0.01)
PFN (Synthetic HW)	0.918 ± (0.005)	0.895 ± (0.02)	0.992 ± (0.003)	0.945 ± (0.005)
PFN (Synthetic HW + ZCP)	**0.918** ± (0.001)	**0.947** ± (0.01)	**0.994** ± (0.002)	**0.953** ± (0.004)

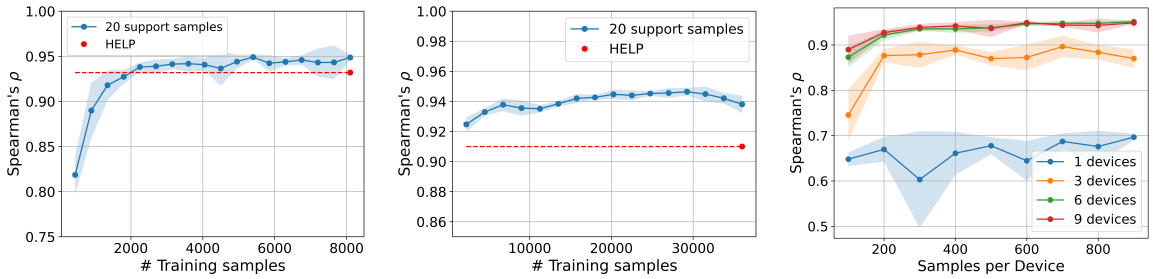
devices except for the CPU. Here, we achieved a Spearman's ρ of 0.983, which is a 0.6% drop in performance to *HELP*'s 0.989.

FBNet performance We ran the same experiments as above for the FBNet search space, the results of which are summarised in Table 5 (see Appendix D). Just as we observed for the NB201 search space, our model improved consistently as we increase the complexity of our priors. Moreover, the performance uplift over *HELP* was even more pronounced across all three test devices as we improved by 4.5% (0.951 vs. 0.91). In this case, our model with the Synthetic HW + ZCP was better than *HELP* across all test devices.

3.2 Sample efficiency

Since acquiring latency samples is expensive, we also evaluate the sample efficiency of our method by varying the number of training samples. For this, we consider two settings: (1) varying the total number of training samples and (2) additionally varying the number of hardware devices in the training set. For the latter, we argue that hardware device efficiency is much more beneficial since adding a new device to one's data collection pipeline is generally much more time intensive and expensive than acquiring more latency samples.

For the first scenario, the results are shown in Figure 1. We observe that our model trained on the Synthetic HW prior outperforms *HELP* with only 22% of the samples. For FBNet we outperform *HELP* for all tested number of samples which equates to a 92% improvement in sample efficiency in the best case.



(a) NB201: sample efficiency (b) FBNet: sample efficiency (c) NB201: device efficiency

Figure 1: Model performance as a function of the number of samples and devices used during training when using the Synthetic HW prior. **(a)** Nine devices on NB201 with training samples ranging from 450 to 8, 100 in 450 sample increments. **(b)** Nine devices on FBNet with training samples ranging from 2, 250 to 36, 000 in 2, 250 sample increments. **(c)** NB201 with varying number of training devices (1 to 9) and samples per device (100 to 900).

In the second scenario, we observe that training with either six or nine devices results in near identical performance (Figure 1c). However, there is quite a drastic drop in performance for any fewer devices used. Though, our model outperforms *HELP* when using only six devices and 300 training samples per device which equates to a 77.8% improvement in sample efficiency.

4 Conclusion

We propose a latency surrogate based on in-context learning that is trained on real data points with data augmentation. We show that our method is effective, outperforming the current state of the art w.r.t. prediction quality and sample efficiency.

5 Limitation

The main limitation of our method is that one needs to retrain the entire model when moving to a different search space. Future work may improve upon our method by investigating the use of different neural architecture encodings that may facilitate such a transfer.

6 Broader Impact Statement

Our method helps at making HW-NAS more efficient, and hence, more accessible, pushing towards democratization of AI, which is the overarching goal of AutoML. This makes it easier to automatically design hardware-efficient deep learning models and, hence, potentially amplifies the societal impacts – both positive and negative – of deep learning.

Acknowledgements. Robert Bosch GmbH is acknowledged for financial support.

References

Abdelfattah, M. S., Mehrotra, A., Dudziak, L., and Lane, N. D. (2021). Zero-cost proxies for lightweight NAS. In *International Conference on Learning Representations (ICLR)*.

Akhauri, Y. and Abdelfattah, M. S. (2023). Multi-predict: Few shot predictors for efficient neural architecture search. In *AutoML Conference 2023*.

Benmeziane, H., El Maghraoui, K., Ouarnoughi, H., Niar, S., Wistuba, M., and Wang, N. (2021). Hardware-aware neural architecture search: Survey and taxonomy. In *IJCAI*, pages 4322–4329.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. (2019). Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*.

Cai, H., Zhu, L., and Han, S. (2018). ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*.

Dudziak, L., Chau, T., Abdelfattah, M., Lee, R., Kim, H., and Lane, N. (2020). Brp-nas: Prediction-based NAS using GCNs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 10480–10490.

Jaitly, N. and Hinton, G. E. (2013). Vocal tract length perturbation (vtlp) improves speech recognition. In *Proc. ICML Workshop on Deep Learning for Audio, Speech and Language*, volume 117, page 21.

Krishnakumar, A., White, C., Zela, A., Tu, R., Safari, M., and Hutter, F. (2022). Nas-bench-suite-zero: Accelerating research on zero cost proxies. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 28037–28051.

Lee, H., Lee, S., Chong, S., and Hwang, S. J. (2021). Hardware-adaptive efficient latency prediction for NAS via meta-learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Lin, M., Wang, P., Sun, Z., Chen, H., Sun, X., Qian, Q., Li, H., and Jin, R. (2021). Zen-nas: A zero-shot NAS for high-performance image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 347–356.

Lopes, V., Alirezazadeh, S., and Alexandre, L. A. (2021). Epe-nas: Efficient performance estimation without training for neural architecture search. In *International conference on artificial neural networks*, pages 552–563. Springer.

Mellor, J., Turner, J., Storkey, A., and Crowley, E. J. (2021). Neural architecture search without training. In *International Conference on Machine Learning (ICML)*, pages 7588–7598. PMLR.

Müller, S., Hollmann, N., Arango, S. P., Grabocka, J., and Hutter, F. (2022). Transformers can do bayesian inference. In *International Conference on Learning Representations (ICLR)*.

Ning, X., Tang, C., Li, W., Zhou, Z., Liang, S., Yang, H., and Wang, Y. (2021). Evaluating efficient performance estimators of neural architectures. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 12265–12277.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning (ICLR)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104. PMLR.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *International Conference on Machine Learning (ICML)*, pages 2902–2911. PMLR.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 6377–6389.

Turner, J., Crowley, E. J., O'Boyle, M., Storkey, A., and Gray, G. (2020). Blockswap: Fisher-guided block substitution for network compression on a budget. In *International Conference on Learning Representations (ICLR)*.

Wang, C., Zhang, G., and Grosse, R. (2020). Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations (ICLR)*.

White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., and Hutter, F. (2023). Neural Architecture Search: Insights from 1000 Papers. arXiv:2301.08727 [cs, stat].

Zhang, L. L., Han, S., Wei, J., Zheng, N., Cao, T., Yang, Y., and Liu, Y. (2021). nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, page 81–93, New York, NY, USA. ACM.

Submission 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] [See Experiments sectiom]

   (b) Did you describe the limitations of your work? [Yes] [See Limitations]

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] [See Broader Impact Statement]

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? `https://2022.automl.cc/ethics-accessibility/` [Yes]

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? [Yes] [We compared all methods using the same evaluation protocol and data]

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? [No] [space constraints]

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] [Across 5 different seeds]

   (d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? [Yes] [We report mean and standard deviation across seeds]

   (e) Did you report the statistical significance of your results? [No]

   (f) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] [We used the ZCP measurements from NAS-Bench-Suite-Zero.]

   (g) Did you compare performance over time and describe how you selected the maximum duration? [N/A]

   (h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]

   (i) Did you run ablation studies to assess the impact of different components of your approach? [Yes] [We ran an ablation on sample efficiency]

3. With respect to the code used to obtain your results...

   (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 versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [No]

   (b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [N/A]

   (c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [N/A]

   (d) Did you include the raw results of running your experiments with the given code, data, and instructions? [N/A]

(e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [N/A]

4. If you used existing assets (e.g., code, data, models)…

   (a) Did you cite the creators of used assets? [Yes]

   (b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A]

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

5. If you created/released new assets (e.g., code, data, models)…

   (a) Did you mention the license of the new assets (e.g., as part of your code submission)? [N/A]

   (b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [N/A]

6. If you used crowdsourcing or conducted research with human subjects…

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

7. If you included theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

A Latency Data

Table 2: Hardware devices with batch sizes, device type and search space. We use the same hardware devices as in HELP (Lee et al., 2021). The search spaces for which we have latency data are also the same as in HELP (Lee et al., 2021).

Device (batch size)	Type	NAS-Bench-201	FBNet
1080ti (1)	GPU	✓	✓
1080ti (32)	GPU	✓	✓
1080ti (64)	GPU	✗	✓
1080ti (256)	GPU	✓	✗
2080ti (1)	GPU	✓	✓
2080ti (32)	GPU	✓	✓
2080ti (64)	GPU	✗	✓
2080ti (256)	GPU	✓	✗
Titan RTX (1)	GPU	✓	✓
Titan RTX (32)	GPU	✓	✓
Titan RTX (64)	GPU	✗	✓
Titan RTX (256)	GPU	✓	✗
Titan X (1)	GPU	✓	✓
Titan X (32)	GPU	✓	✓
Titan X (64)	GPU	✗	✓
Titan X (256)	GPU	✓	✗
Titan XP (1)	GPU	✓	✓
Titan XP (32)	GPU	✓	✓
Titan XP (64)	GPU	✗	✓
Titan XP (256)	GPU	✓	✗
Intel Xeon Gold 6266 (1)	CPU	✓	✓
Intel Xeon Gold 6240 (1)	CPU	✓	✓
Intel Xeon Silver 4114 (1)	CPU	✓	✓
Intel Xeon Silver 4210r (1)	CPU	✓	✓
essential phone 1 (1)	Mobile	✓	✓
Pixel 2 (1)	Mobile	✓	✓
Pixel 3 (1)	Mobile	✓	✓
Samsung A50 (1)	Mobile	✓	✓
Samsung S7 (1)	Mobile	✓	✓
eyeriss (1)	ASIC	✓	✓
fpga (1)	FPGA	✓	✓
Raspi 4 (1)	Embedded	✓	✓

Table 3: Hardware-device data splits (batch sizes, if more than 1 is available and used) following HELP (Lee et al., 2021)

Split	NAS-Bench-201	FBNet
Training	1080ti (1, 32, 256), Silver 4144, Silver 4210r, Samsung A50, Pixel 3, Essential Phone 1, Samsung S7	1080ti (1, 32, 64), Silver 4114, Silver 4210r, Samsung A50, Pixel 3, Essential Phone 1, Samsung S7
Validation	Titan X (1, 32, 256), Gold 6240	Titan X (1, 32, 64), Gold 6240
Test	Titan RTX (256), Gold 6226, FPGA, Pixel 2, Raspi 4, Eyeriss	FPGA, Raspi 4, Eyeriss

B Neural Architecture Encoding

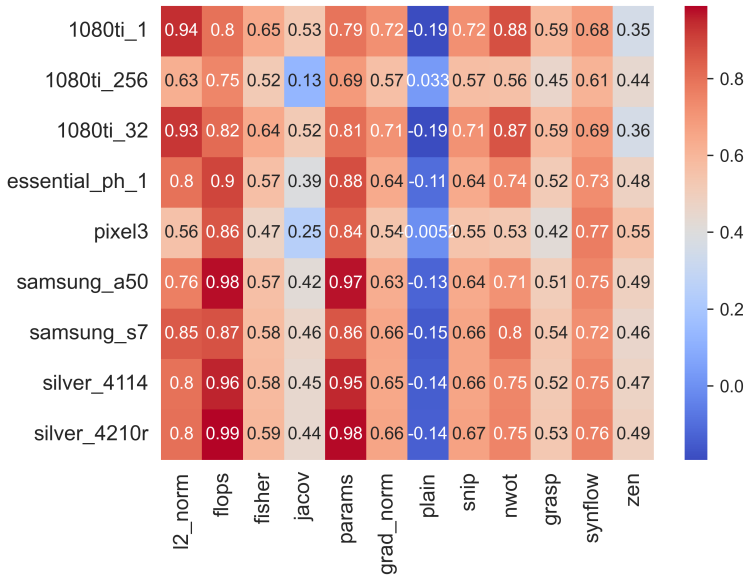


Figure 2: Spearman's ρ of hardware-latencies of all architectures in NAS-Bench-201 for the training hardware-platforms used in HELP (Lee et al., 2021) and ZCPs used in Multi-Predict (Akhauri and Abdelfattah, 2023).

C Zero Cost Proxies

Table 4: Zero Cost Proxies used to represent the architectures in the NAS-Bench-201 and FBNet search spaces. We primarily use the same ZCPs as in Akhauri and Abdelfattah (2023) except for zen-score.

ZCPs	Type	NAS-Bench-201	FBNet
fisher (Turner et al., 2020)	Pruning-at-init	✓	✓
grad-norm (Abdelfattah et al., 2021)	Pruning-at-init	✓	✓
grasp (Wang et al., 2020)	Pruning-at-init	✓	✓
snip (Abdelfattah et al., 2021)	Pruning-at-init	✓	✓
synflow (Tanaka et al., 2020)	Pruning-at-init	✓	✓
synflow-bn (Tanaka et al., 2020)	Pruning-at-init	✗	✓
FLOPs (Ning et al., 2021)	Baseline	✓	✗
Params (Ning et al., 2021)	Baseline	✓	✓
MACs (Lee et al., 2021)	Baseline	✗	✓
plain (Abdelfattah et al., 2021)	Baseline	✓	✓
l2-norm (Ning et al., 2021)	Baseline	✓	✓
jacov (Mellor et al., 2021)	Jacobian	✓	✓
nwot (Mellor et al., 2021)	Jacobian	✓	✓
epe-nas (Lopes et al., 2021)	Jacobian	✗	✓
zen-score (Lin et al., 2021)	Piecewise Linear	✓	✗

D FBNet results

Table 5: Spearman's ρ for predicted vs. actual latencies on the FBNet search space with 20 support datapoints.

Method		Unseen Device			Mean
		Raspi4	Eyeriss	FPGA	
HELP (Lee et al., 2021)		0.885	0.942	0.889	0.91
PFN (Ours)	no-prior	0.805 ± (0.09)	0.854 ± (0.09)	0.798 ± (0.06)	0.819 ± (0.08)
	+ latency scaling	0.877 ± (0.02)	0.953 ± (0.02)	0.902 ± (0.01)	0.911 ± (0.01)
	+ lin. comb. (Synthetic HW)	0.920 ± (0.01)	0.957 ± (0.006)	0.935 ± (0.01)	0.937 ± (0.01)
	+ lin. comb. (ZCP)	**0.926** ± (0.01)	**0.971** ± (0.002)	**0.955** ± (0.004)	**0.951** ± (0.004)