A SUPPLEMENTAL MATERIAL

A.1 Best practices for NAS

(White et al., 2020; Li & Talwalkar, 2019; Ying et al., 2019; Yang et al., 2020) discuss improving reproducibility and fairness in experimental comparisons for NAS. We thus address the sections released in the NAS best practices checklist by (Lindauer & Hutter, 2019).

- **Best Practice: Release Code for the Training Pipeline(s) you use:** We release code for our Predictor, CATE, Arch2Vec encoder training set-up.
- **Best Practice: Release Code for Your NAS Method:** We do not conduct NAS.
- **Best Practice: Use the Same NAS Benchmarks, not Just the Same Datasets:** We use the NASBench-201 and FBNet datasets for evaluation. We also use a subset of Zero Cost Proxies from NAS-Bench-Suite-Zero.
- **Best Practice: Run Ablation Studies:** We run extensive ablation studies in our paper. We conduct ablation studies with different supplementary encodings in the main paper as well as predictor ablations in the appendix.
- **Best Practice: Use the Same Evaluation Protocol for the Methods Being Compared:** We use the same evaluation protocol as existing works in this field (HELP (Lee et al., 2021b) and MultiPredict (Akhauri & Abdelfattah, 2023))
- **Best Practice: Evaluate Performance as a Function of Compute Resources:** In this paper, we study the sample efficiency of latency predictors. We report results in terms of the 'number of trained models required'. This directly correlates with compute resources, depending on the NAS space training procedure.
- **Best Practice: Compare Against Random Sampling and Random Search:** We propose a end-to-end predictor design methodology, not a NAS method.
- **Best Practice: Perform Multiple Runs with Different Seeds:** Our appendix contains information on number of trials and our tables in the main paper are with standard deviation.
- **Best Practice: Use Tabular or Surrogate Benchmarks If Possible:** All our evaluations are done on publicly available Tabular and Surrogate benchmarks.

Table 9. In our tests, cosine based selection for samplers with vector encodings outperforms kMeans. Tested on Task N3 with operation-wise hardware embedding and hardware embedding initialization.

	10 Samples			
NB201	ZCP	Arch2Vec	CATE	CAZ
Cosine	**0.948**	**0.949**	**0.933**	**0.951**
Kmeans	0.729	0.670	0.826	0.892
FBNet	ZCP	Arch2Vec	CATE	CAZ
Cosine	**0.822**	**0.803**	**0.805**	**0.788**
Kmeans	0.412	0.657	NaN	0.718
	20 Samples			
NB201	ZCP	Arch2Vec	CATE	CAZ
Cosine	**0.963**	**0.960**	**0.952**	**0.960**
Kmeans	0.786	0.680	0.885	0.948
FBNet	ZCP	Arch2Vec	CATE	CAZ
Cosine	**0.845**	**0.839**	**0.828**	**0.852**
Kmeans	0.818	0.812	NaN	0.835

A.2 Experimental settings for tables

Table 2: Random seeds are used to generate 4 different device sets for NB201 and FBNet each.
Table 1 uses Random sampler without supplementary encoding, 20 samples on the target device. No supplementary encoding is used.
Table 3: only 5 samples on the test device are used for transfer to effectively test different samplers under few-shot conditions. No supplementary encoding is used.
Table 4: CAZ + kMeans was used as the sampler and 20 samples are fetched for transfer to ensure effective training of baseline predictor.
Table 5: 20 samples are used for transfer with random sampler, no supplementary encoding.
Table 7: Predictor is trained with the CAZ and CATE sampler with ZCP and Arch2Vec supplemental encoding for NASBench-201 and FBNet respectively.
Table 6: ZCP and Arch2Vec supplemental encoding, CAZ and CATE sampler for NASBench-201 and FBNet respectively. 20 samples used for transfer to target device.

A.3 NAS Search Spaces

Latency predictors for neural architecture search (NAS) generally operate on a pre-defined search space of neural network architectures. Several such architecture search spaces can be represented as cells where nodes represent activations and edges represent operations. In this paper, we evaluate a wide range of hardware devices across 11 representative platforms described in Table 2 on two neural

Figure 4. Standard deviation of neural network samplers using supplementary encodings are generally lower than random methods.

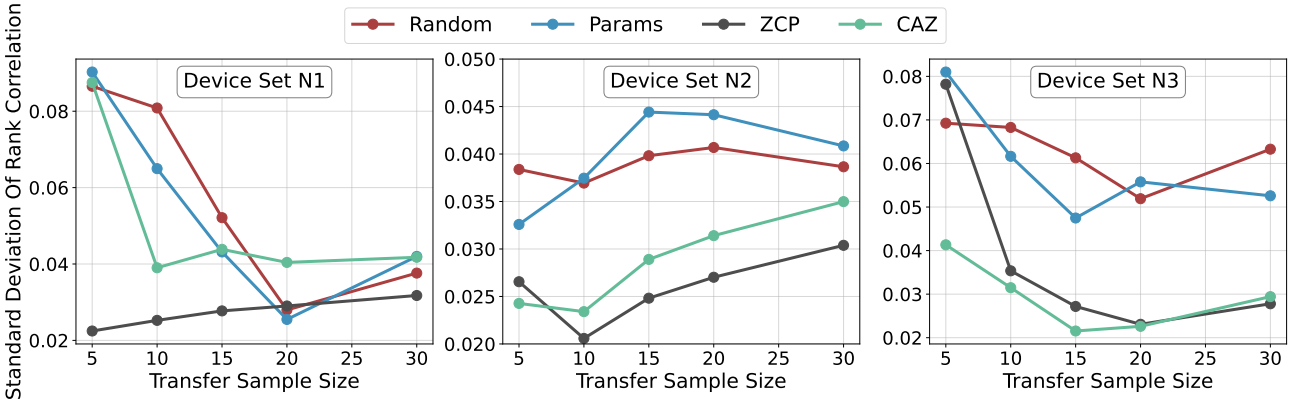
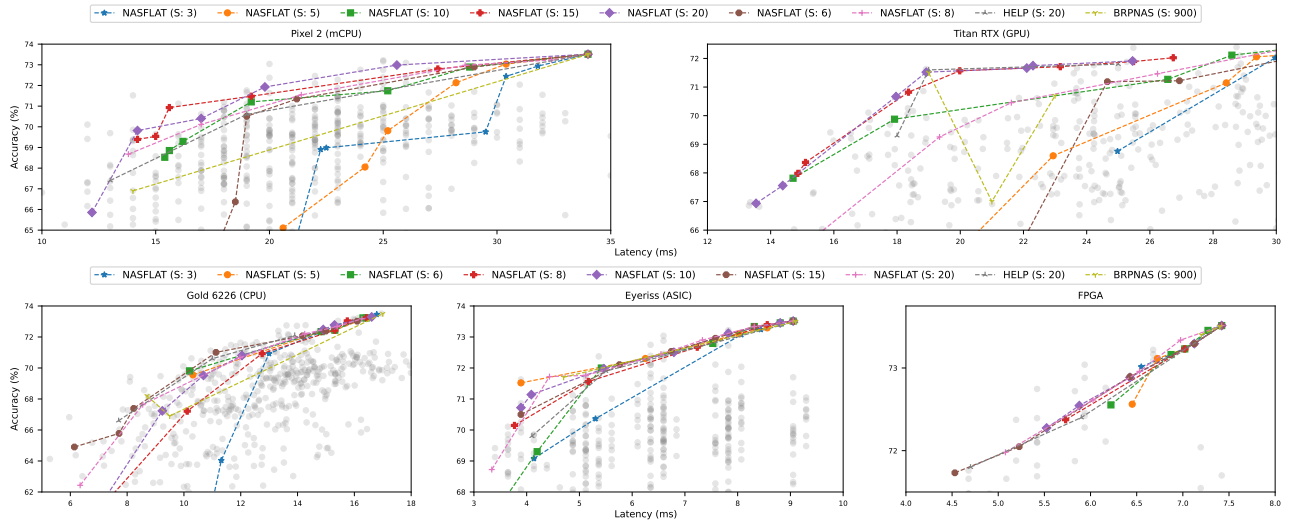


Figure 5. Latency-Accuracy NAS results for different sample sizes (S)



architecture search spaces, NASBench-201 and FBNet. The hardware latency data-set generated for our tests is collated from HW-NAS-Bench (Li et al., 2021) and Eagle (Dudziak et al., 2020).

Micro Cell Space (NASBench-201) (Dong & Yang, 2020) is a cell-based architecture design with each cell comprising 4 nodes and 6 edges. Edges can have one of five types: zero-rize, skip-connection, 1×1 convolution, 3×3 convolution, or 3×3 average-pooling. With 15625 unique architectures, the entire network is assembled using a stem cell, three stages of five cell repetitions, followed by average-pooling and a final softmax layer.

Macro Cell Space (FBNet) (Wu et al., 2019) features a fixed macro architecture with a layer-wise search space. It offers 9 configurations of 'candidate blocks' across 22 unique positions, leading to approximately 10^{21} potential architectures. Despite its macro nature, FBNet can be cell-represented with 22 operational edges. For consistency, we model both NASBench-201 and FBNet using adjacency and operation matrices.

A.3.1 GNN Module Design

Despite the improved performance that GCNs can deliver, they suffer from an over-smoothing problem (Ming Chen et al., 2020), where this is a gradual loss of discriminative information between nodes due to the convergence of node features across multiple aggregation layers. To this end, GATES (Ning et al., 2023; 2022) introduced a custom GCN module referred to as Dense Graph Flow (DGF), which utilizes residual connections within the DGF to improve performance. Additionally, we study another node propagation mechanism based on graph attention.

Dense Graph Flow (DGF): The Dense Graph Flow (DGF) module implements residual connections to retain localized, discriminative features. To describe this mathematically, consider X^l as the input feature matrix for layer l, A as the

adjacency matrix, and O as the operator embedding. The corresponding parameters and bias terms for this layer are represented by W_o^l, W_f^l, and b_f^l. Using these, the input feature matrix for the subsequent layer, X^{l+1}, is determined using the sigmoid activation function, σ, as:

$$X^{l+1} = \sigma(OW_o^l) \odot (AX^lW_f^l) + X^lW_f^l + b_f^l \quad (1)$$

Graph Attention (GAT): The GAT approach (Veličković et al., 2018) distinguishes itself from DGF by its attention mechanism during node information aggregation. Rather than utilizing a linear transformation W_o^l like in DGF for operation features, GAT assesses pairwise interactions among nodes via its dedicated attention layer. For layer l, node features (or input feature matrix) are denoted by X^l. A linear transformation characterized by the projection matrix W_p^l uplifts the input to advanced features. Subsequently, node features undergo self-attention via a common attentional mechanism, denoted as a. S refers to SoftMax and L refers to LeakyReLU. Hence, the output X^{l+1} is formulated as:

$$\text{Attn}_j(X^l) = \text{S}(\text{L}(A_j \cdot a(W_p^lX^l \odot W_pX_j^l))) \odot W_pX_j^l \quad (2)$$

$$X^{l+1} = \text{LayerNorm}\left(\sigma(OW_o^l) \odot \sum_{j=1}^{n} \text{Attn}_j(X^l)\right) \quad (3)$$

Here, Attn_j represents the normalized attention coefficients, and σ is the sigmoid activation function. To enhance GAT's efficacy, we integrate the learned operation attention mechanism W_o (as mentioned in Equation 1) with pairwise attention. This combined attention scheme fine-tunes the aggregated information. Additionally, we incorporate LayerNorm to ensure training stability.

A.4 Predictor Design Ablation

In this subsection, we conduct an in-depth study of predictor design inspired by recent research, but from an accuracy maximization perspective. We use our ablation on accuracy to design a state-of-the-art predictor that we use for our latency study. To achieve a fair evaluation, we test each design improvement on a huge set of neural architecture design spaces detailed below.

A.4.1 Neural Architecture Design Spaces

In this study, we examine various unique neural architecture design domains. We delve into NASBench-101 (Ying et al., 2019) and NASBench-201 (Dong & Yang, 2020), both of which are cell-based search spaces encompassing 423,624 and 15,625 architectures, respectively. While NASBench-101 is trained on CIFAR-10, NASBench-201 benefits from training on CIFAR-10, CIFAR-100, and ImageNet16-120. NASBench-301 (Zela et al., 2020) acts as a surrogate NAS benchmark with an impressive count of 10^{18} architectures. Meanwhile, TransNAS-Bench-101 (Duan et al., 2021) offers both a micro (cell-based) search area featuring 4096 architectures and a broader macro search domain with 3256 designs. For the scope of our study, we focus on the TransNASBench-101 Micro due to its cell-based nature. Each of these networks undergoes training across seven distinct tasks sourced from the Taskonomy dataset. The NASLib framework brings coherence to these search areas. NAS-Bench-Suite-Zero (Krishnakumar et al., 2022) expands the landscape by introducing two datasets from NAS-Bench-360 and four more from Taskonomy. It's worth noting that the NDS dataset features '$FixWD$' datasets, signifying that the architectures maintain consistent width and depth.

A.4.2 Training analogous predictor training

Given a DAG, the TA-GATES(Ning et al., 2022) encoding begins by obtaining the initial operation embedding for all operations based on their types. For T time steps, an iterative process updates the operation embeddings, mimicking architecture parameter updates in training. Each step involves computing information flow via GCNs on the architecture DAG, and calling an MLP using the previous operation embedding; the output is then used in a backward GCN pass and then computing the operation embedding update. The last step concatenates the previous operation embedding with the forward and backward propagated information, feeds this to an MLP, and yields the updated operation embedding for the next step. The final architecture encoding uses the output of the T-th iterative refinement. In Figure 6 we study the impact of changing the number of time-steps on the over-all kendall tau correlation (KDT). The trend is inconsistent, and therefore we attempt to investigate the utility of the backward GCN.

To conduct a deeper investigation of this phenomenon, we study several aspects of training analogous predictor training. We look at `BMLP`, which is where we replace the backward 'GCN' with a small MLP. Further, the backward GCN pass uses the output of the 'forward' GCN along with the transposed adjacency matrix. This is then passed to an 'operation update MLP' that takes as input `BYI`, which is the output of the backward GCN, `BOpE`, which is the operation embedding itself. In Table 14, Table 12, Table 15, Table 13 we can see that in all cases, `BMLP` outperforms having a backward GCN. Additionally, in many cases the `BYI` information does not add much value. However, it does not harm performance. Therefore, for further tests we will use `BMLP` with `BYI,BOpE`.

Space	Amoeba		DARTS		ENAS		ENAS_fix-w-d		NASNet		PNAS		nb101		nb201	
All Node Encoding	False	True	False	True	False	True	False	True	False	True	False	True	False	True	False	True
Samples																
8	0.100	0.078	0.079	0.076	0.075	0.078	0.183	0.154	0.135	0.122	0.082	0.089	0.395	0.370	0.445	0.533
16	0.202	0.157	0.178	0.184	0.165	0.186	0.322	0.302	0.155	0.127	0.124	0.127	0.448	0.350	0.647	0.656
32	0.287	0.293	0.246	0.251	0.295	0.277	0.319	0.301	0.223	0.223	0.246	0.251	0.543	0.532	0.709	0.715
64	0.375	0.367	0.416	0.411	0.364	0.357	0.374	0.372	0.347	0.331	0.348	0.330	0.652	0.623	0.771	0.762
128	0.455	0.419	0.476	0.474	0.463	0.442	0.386	0.388	0.432	0.398	0.505	0.494	0.703	0.698	0.818	0.808

Table 10. Investigating the impact of whether we should use the features of **every** node for the backward MLP (True) or not (False).

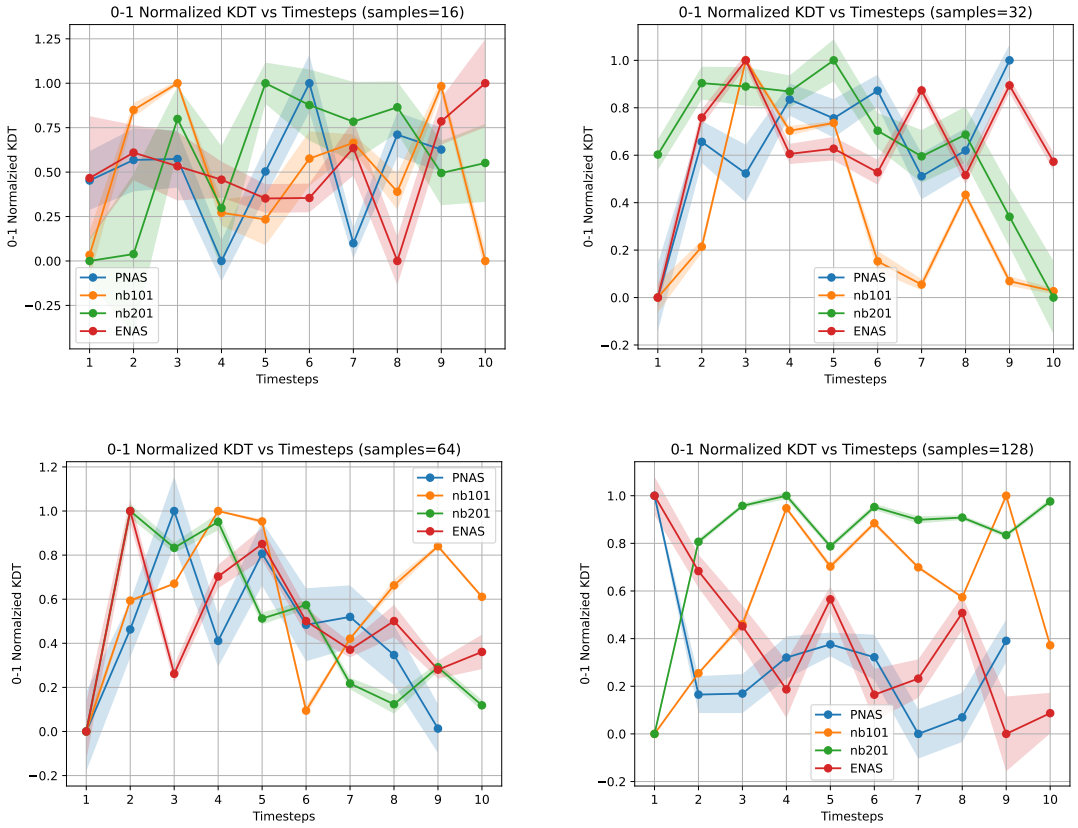


Figure 6. Testing the impact of time-steps in the training-analogous operation update regime. 0-1 normalized KDT for each space.

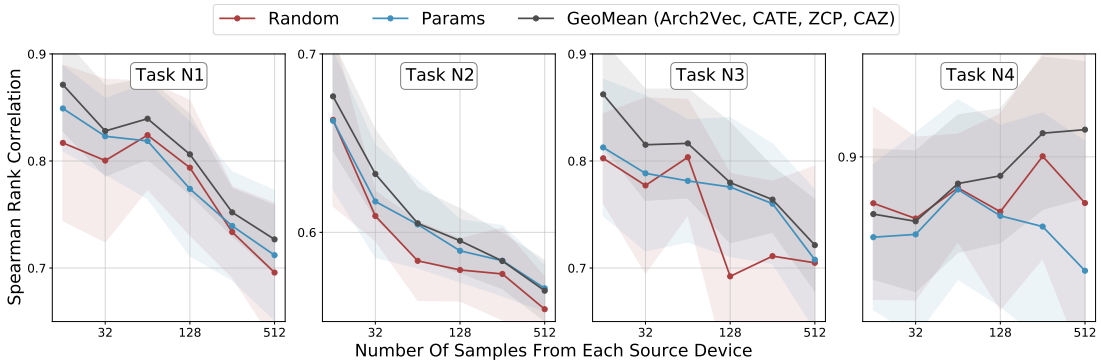
Table 11. Sample Size 64 (left), 128 (right)

Space	DOpEmbUnrolled BMLP	Default	DOpEmbUnrolled GCN	DOpEmbUnrolled BMLP	Default	DOpEmbUnrolled GCN
PNAS	0.3244	0.3	0.3395	0.4779	0.4684	0.4481
ENAS	0.409	0.3929	0.3764	0.4925	0.4958	0.4498
nb201	0.7831	0.7795	0.7640	0.7953	0.8007	0.7838
nb101	0.6055	0.6283	0.6041	0.7122	0.7013	0.6993

A.4.3 Inputs to backward MLP and gradient flow

From Figure 6, we have observed that 2 timesteps generally help but more timesteps are not useful for predictor performance. Additionally, we have replaced the entire backward GCN with a backward MLP. We now investigate which gradients need to be detached during iterative refinement. In the 'def' (default) TA-GATES case, the `BYI` is **not detached**, whereas the `BOpE` is detached. In our tests, in 'all', we detach `BYI`, `BOpE` and in 'none', we do not detach any inputs to the `BMLP`. We find no clear pattern over 8 tests (3 trials each) in detach, except that it is better to either use the

Figure 7. More latency samples from training (source) devices may degrade predictor performance for difficult latency prediction tasks. We use 20 samples for the target device.



default rule or detach none of the gradients. For simplicity, we will detach none of the gradients. We again see that `BYI` is important for the `BMLP`, but the utility of `BOpE` is unclear. Further, In Table 10, we test whether we need to pass only the encoding at the output node of the forward GCN, or should we concatenate encodings at all nodes to pass to the backward MLP/GCN. Here, we find that there is no clear advantage of passing all node encodings and thus we only use the output node encoding.

A.4.4 Unrolled backward MLP computation

Here, we investigate different ways to unroll the backward computation to simplify the encoding process even further. In Table 11, we introduce two methods of unrolling the computation. We only unroll for 2 time-steps, which gives us a computational graph very similar to Figure 3. In the case of `DOpEmbUnrolled BMLP`, (Direct Op-Emb Unrolled to BMLP) we directly take the output of the forward GNN along with the operation embedding, pass it to an MLP and use that as the encoding for the next GNN. In the case of `DOpEmbUnrolled GCN` (Direct Op-Emb Unrolled to GCN) we directly take the output of the forward GNN along with the operation embedding and pass it to the backward-GCN instead of the BMLP, and use the output as an encoding for the next GNN. We find that unrolling the computation further improves predictive performance.

A.4.5 Final Predictor Architecture Design

Finally, this leads us to our own architecture design. In our architecture, we significantly simplify the predictor architecture. Firstly, we maintain a smaller GNN which refines the operation and hardware embedding. This refind embedding is passed to an MLP which maps the embedding back to the original dimensions. This refined embedding is passed directly to the larger GNN along with the adjacency matrix and node embeddings. We find that this simplified architecture performs better in most of our tests.

Table 12. ENAS Sample Size 64 (left), 128 (right). Ablation for backward pass. BMLP indicates that instead of replicating network, we do a simple 2 layer MLP for backward pass. BYI indicates whether we use the output of the forward pass network or not. BOpE indicates whether we use the output of the operation embedding itself or not. PM indicates that num params are approximately matched wrt TS > 1. w / d indicates whether the matching happens by adjusting width or depth in forward gcn architecture. 2R implies that we simply use a small random perturb vector for operation update to 'regularize' the network.

TS	BMLP	BYI	BOpE	KDT	Dev
2R	✗	✗	✗	0.3397	0.0018
1^d_{PM}	✗	✗	✗	0.3832	0.0027
2	✓	✓	✗	0.3941	0.0061
3	✓	✗	✓	0.3973	0.0008
1^w_{PM}	✗	✗	✗	0.3983	0.006
3	✓	✓	✗	0.3988	0.0054
2	✓	✗	✓	0.4142	0.0009

TS	BMLP	BYI	BOpE	KDT	Dev
2R	✗	✗	✗	0.4611	0.0010
1^w_{PM}	✗	✗	✗	0.4635	0.0027
1^d_{PM}	✗	✗	✗	0.4684	0.0007
2	✓	✗	✗	0.4686	0.0005
3	✗	✓	✗	0.4847	0.0007
3	✓	✗	✓	0.4888	0.0039
2	✓	✗	✓	0.4975	0.0021

Table 13. NB201 Sample Size 64 (left), 128 (right). Ablation for backward pass.

TS	BMLP	BYI	BOpE	KDT	Dev
1^d_{PM}	✗	✗	✗	0.7472	0.0005
1^w_{PM}	✗	✗	✗	0.7475	0.0005
2R	✗	✗	✗	0.7521	0.0002
3	✓	✗	✗	0.7835	0.0002
2	✓	✓	✓	0.7845	0.0001
3	✓	✗	✓	0.7856	0.0001
3	✓	✓	✓	0.7882	0.0003
2	✓	✗	✓	0.7894	0.0002

TS	BMLP	BYI	BOpE	KDT	Dev
1^w_{PM}	✗	✗	✗	0.7682	0.0
2R	✗	✗	✗	0.7718	0.0001
1^d_{PM}	✗	✗	✗	0.7765	0.0005
2	✓	✗	✓	0.7918	0.0003
2	✓	✓	✓	0.7927	0.0004
3	✓	✓	✓	0.7953	0.0005
2	✓	✓	✗	0.7956	0.0001
3	✓	✗	✓	0.7971	0.0004

Table 14. PNAS Sample Size 64 (left), 128 (right). Ablation for backward pass.

TS	BMLP	BYI	BOpE	KDT	Dev
1^d_{PM}	✗	✗	✗	0.3387	0.008
2R	✗	✗	✗	0.3494	0.0165
1^w_{PM}	✗	✗	✗	0.3635	0.011
2	✗	✓	✗	0.364	0.0145
2	✓	✗	✓	0.3641	0.0055
3	✗	✓	✗	0.378	0.0106
2	✓	✓	✗	0.382	0.0092
3	✓	✓	✗	0.3852	0.0104

TS	BMLP	BYI	BOpE	KDT	Dev
1^d_{PM}	✗	✗	✗	0.4352	0.012
2R	✗	✗	✗	0.4507	0.0063
2	✓	✗	✗	0.4625	0.0035
2	✓	✓	✗	0.4684	0.0033
3	✓	✓	✗	0.4709	0.0041
1^w_{PM}	✗	✗	✗	0.4779	0.003
3	✗	✓	✗	0.4841	0.0005
2	✗	✓	✗	0.4897	0.0006

Table 15. NB101 Sample Size 64 (left), 128 (right). Ablation for backward pass.

TS	BMLP	BYI	BOpE	KDT	Dev
1^d_{PM}	✗	✗	✗	0.6211	0.0007
1^w_{PM}	✗	✗	✗	0.6273	0.0003
2	✓	✗	✗	0.6346	0.0001
2R	✗	✗	✗	0.6421	0.0063
2	✓	✓	✗	0.6466	0.0008
2	✓	✗	✓	0.6502	0.0004
3	✓	✓	✗	0.6515	0.0006
2	✓	✓	✓	0.6537	0.0003

TS	BMLP	BYI	BOpE	KDT	Dev
1^w_{PM}	✗	✗	✗	0.6591	0.0003
2R	✗	✗	✗	0.6886	0.0063
1^d_{PM}	✗	✗	✗	0.7008	0.0001
2	✓	✓	✗	0.707	0.0004
2	✓	✓	✓	0.7075	0.0001
2	✓	✗	✗	0.7089	0.0002
2	✓	✗	✓	0.7194	0.0001
3	✓	✓	✗	0.7276	0.0002

Table 16. ENAS Sample Size 64 (left), 128 (right). Ablation for backward pass. BMLP is always True. BYI indicates whether we use the output of the forward pass network or not. BOpE indicates whether we use the output of the operation embedding itself or not. DM indicates detachment mode. 2 timesteps fixed.

BYI	BOpE	DM	KDT	STD	BYI	BOpE	DM	KDT	STD
✓	✗	def	0.4012	0.0062	✓	✓	none	0.4694	0.0025
✓	✗	none	0.4042	0.0044	✗	✗	none	0.4716	0.0005
✗	✓	none	0.4058	0.0013	✗	✓	all	0.4958	0.003
✓	✗	all	0.4074	0.0053	✗	✓	def	0.4975	0.0021
✗	✓	def	0.4142	0.0009	✗	✓	none	0.511	0.0018

Table 17. NB201 Sample Size 64 (left), 128 (right). Ablation for backward pass.

BYI	BOpE	DM	KDT	STD	BYI	BOpE	DM	KDT	STD
✗	✓	def	0.7795	0.0001	✓	✓	def	0.7875	0.0003
✓	✗	none	0.7853	0.0002	✓	✗	none	0.7888	0.0002
✓	✓	def	0.7859	0.0003	✓	✓	none	0.7936	0.0005
✗	✓	none	0.7871	0.0002	✗	✓	none	0.7944	0.0002
✓	✓	none	0.7949	0.0004	✗	✓	def	0.8007	0.0002

Table 18. PNAS Sample Size 64 (left), 128 (right). Ablation for backward pass.

BYI	BOpE	DM	KDT	STD	BYI	BOpE	DM	KDT	STD
✓	✓	def	0.3184	0.0011	✓	✗	none	0.4388	0.0037
✓	✗	none	0.3304	0.0067	✓	✗	def	0.456	0.0036
✓	✗	all	0.333	0.006	✗	✓	none	0.4675	0.0039
✓	✗	def	0.3459	0.0074	✗	✓	all	0.4684	0.0042
✓	✓	none	0.3538	0.0045	✗	✓	def	0.474	0.0042

Table 19. NB101 Sample Size 64 (left), 128 (right). Ablation for backward pass.

BYI	BOpE	DM	KDT	STD	BYI	BOpE	DM	KDT	STD
✓	✓	none	0.6329	0.0006	✗	✗	def	0.7139	0.0001
✓	✓	all	0.6432	0.0014	✓	✓	all	0.7177	0.0001
✗	✓	none	0.6436	0.0013	✗	✗	none	0.7206	0.0002
✗	✗	all	0.6552	0.0	✓	✗	def	0.728	0.0
✓	✓	def	0.6588	0.0	✓	✗	none	0.735	0.0

Hyperparameter	**Value**	**Hyperparameter**	**Value**
Learning Rate	0.001	Weight Decay	0.00001
Number of Epochs	150	Batch Size	16
Number of Transfer Epochs	40 NB201, 30 FBNet	Transfer Learning Rate	0.003 NB201, 0.001 FBNet
Graph Type	`DGF+GAT ensemble`	Op Embedding Dim	48
Node Embedding Dim	48	Hidden Dim	96
Op-HW GCN Dims	[128, 128]	Op-HW MLP Dims	[128]
GCN Dims	[128, 128, 128]	MLP Dims	[200, 200, 200]
Number of Trials	3	Loss Type	Pairwise Hinge Loss (Ning et al., 2022)

Table 20. Hyperparameters used in the experiments. We run Optuna hyper-parameter optimization for 80 iterations.

NASBench201 ND Train-Test Correlation	Latency Correlation between Test and Train devices								
	1080ti_1	1080ti_32	1080ti_256	silver_4114	silver_4210r	samsung_a50	pixel3	essential_ph_1	samsung_s7
titan_rtx_256	0.772	0.792	0.812	0.947	0.982	0.975	0.878	0.897	0.854
gold_6226	0.958	0.956	0.776	0.912	0.927	0.894	0.711	0.898	0.920
fpga	0.828	0.841	0.888	0.943	0.974	0.959	0.872	0.924	0.888
pixel2	0.807	0.817	0.777	0.873	0.894	0.874	0.761	0.856	0.832
raspi4	0.654	0.669	0.735	0.844	0.878	0.875	0.967	0.808	0.758
eyeriss	0.415	0.434	0.893	0.586	0.618	0.625	0.722	0.624	0.521

NASBench201 N1 Train-Test Correlation Latency Correlation between Test and Train devices					
	e_tpu_edge_tpu_int8	eyeriss	m_gpu_sd_675_AD_612_int8	m_gpu_sd_855_AD_640_int8	pixel3
1080ti_1	0.167	0.415	0.594	0.551	0.591
titan_rtx_32	0.127	0.403	0.595	0.547	0.599
titanxp_1	0.163	0.405	0.594	0.551	0.589
2080ti_32	0.174	0.424	0.603	0.560	0.601
titan_rtx_1	0.113	0.362	0.554	0.504	0.547

NASBench201 N2 Train-Test Correlation Latency Correlation between Test and Train devices					
	1080ti_1	1080ti_32	titanx_32	titanxp_1	titanxp_32
e_gpu_jetson_nano_fp16	0.514	0.509	0.517	0.510	0.513
e_tpu_edge_tpu_int8	0.167	0.172	0.171	0.163	0.170
m_dsp_sd_675_HG_685_int8	0.593	0.594	0.599	0.591	0.596
m_dsp_sd_855_HG_690_int8	0.587	0.583	0.592	0.585	0.589
pixel3	0.591	0.611	0.598	0.589	0.607

NASBench201 N3 Train-Test Correlation Latency Correlation between Test and Train devices					
	d_gpu_gtx_1080ti_fp32	e_gpu_jetson_nano_fp16	eyeriss	m_dsp_sd_675_HG_685_int8	m_gpu_sd_855_AD_640_int8
1080ti_1	0.362	0.514	0.415	0.593	0.551
2080ti_1	0.356	0.512	0.405	0.586	0.538
titanxp_1	0.356	0.510	0.405	0.591	0.551
2080ti_32	0.371	0.519	0.424	0.598	0.560
titanxp_32	0.370	0.513	0.423	0.596	0.564

NASBench201 N4 Train-Test Correlation Latency Correlation between Test and Train devices										
	d_cpu_i7_7820x_fp32	e_gpu_jetson_nano_fp32	e_tpu_edge_int8	eyeriss	m_cpuSD_855_kryo_485i8	m_dspSD_675_HG_685i8	m_dspSD_855_HG_690i8	m_gpuSD_675_AD_612i8	m_gpuSD_855_AD_640i8	pixel2
1080ti_1	0.360	0.739	0.167	0.415	0.645	0.593	0.587	0.594	0.551	0.807
2080ti_1	0.353	0.730	0.168	0.405	0.635	0.586	0.581	0.581	0.538	0.791
titan_rtx_1	0.313	0.703	0.113	0.362	0.600	0.547	0.541	0.554	0.504	0.775

NASBench201 NA Train-Test Correlation Latency Correlation between Test and Train devices																	
	titan_rtx_1	titan_rtx_32	titanxp_1	2080ti_1	titanx_1	1080ti_1	titanx_32	titanxp_32	2080ti_32	1080ti_32	gold_6226	samsung_s7	silver_4114	gold_6240	silver_4210r	samsung_a50	pixel2
eyeriss	0.362	0.403	0.405	0.405	0.409	0.415	0.418	0.423	0.424	0.434	0.503	0.521	0.586	0.609	0.618	0.625	0.609
d_gpu_gtx_1080ti_fp32	0.315	0.346	0.356	0.356	0.362	0.362	0.369	0.370	0.371	0.376	0.438	0.450	0.488	0.507	0.511	0.513	0.501
e_tpu_edge_tpu_int8	0.113	0.127	0.163	0.168	0.166	0.167	0.171	0.170	0.174	0.172	0.221	0.246	0.243	0.268	0.256	0.261	0.299

Table 21. NASBench-201 task train-test correlations. HG: Hexagon; AD: Adreno; m: Mobile; SD: Snapdragon; e: Embedded; i8: int8. Rows are test devices, Column headers are training devices.

FBNet FD Train-Test Correlation	Latency Correlation between Test and Train devices								
	1080ti_1	1080ti_32	1080ti_64	silver_4114	silver_4210r	samsung_a50	pixel3	essential_ph_1	samsung_s7
fpga	0.226	0.419	0.605	0.674	0.679	0.865	0.906	0.700	0.713
raspi4	0.256	0.524	0.719	0.641	0.649	0.841	0.957	0.660	0.678
eyeriss	0.247	0.527	0.757	0.624	0.633	0.864	0.976	0.678	0.690

FBNet F1 Train-Test Correlation	Latency Correlation between Test and Train devices				
	2080ti_1	essential_ph_1	silver_4114	titan_rtx_1	titan_rtx_32
eyeriss	0.238	0.678	0.624	0.249	0.442
fpga	0.206	0.700	0.674	0.217	0.350
raspi4	0.241	0.660	0.641	0.251	0.450
samsung_a50	0.310	0.646	0.686	0.317	0.429
samsung_s7	0.293	0.650	0.649	0.312	0.352

FBNet F2 Train-Test Correlation	Latency Correlation between Test and Train devices				
	essential_ph_1	gold_6226	gold_6240	pixel3	raspi4
1080ti_1	0.258	0.207	0.536	0.249	0.256
1080ti_32	0.338	0.241	0.459	0.555	0.524
2080ti_32	0.312	0.214	0.449	0.519	0.492
titan_rtx_1	0.268	0.184	0.536	0.253	0.251
titanxp_1	0.286	0.222	0.568	0.270	0.276

FBNet F3 Train-Test Correlation	Latency Correlation between Test and Train devices				
	essential_ph_1	pixel2	pixel3	raspi4	samsung_s7
1080ti_1	0.258	0.300	0.249	0.256	0.307
1080ti_32	0.338	0.409	0.555	0.524	0.372
2080ti_1	0.240	0.287	0.243	0.241	0.293
titan_rtx_1	0.268	0.296	0.253	0.251	0.312
titan_rtx_32	0.313	0.369	0.471	0.450	0.352

FBNet F4 Train-Test Correlation	Latency Correlation between Test and Train devices									
	1080ti_64	2080ti_1	eyeriss	gold_6226	gold_6240	raspi4	samsung_s7	silver_4210r	titan_rtx_1	titan_rtx_32
1080ti_1	0.439	0.944	0.247	0.207	0.536	0.256	0.307	0.653	0.948	0.846
pixel2	0.496	0.287	0.767	0.747	0.678	0.747	0.629	0.653	0.296	0.369
essential_ph_1	0.414	0.240	0.678	0.670	0.663	0.660	0.650	0.608	0.268	0.313

FBNet FA Train-Test Correlation	Latency Correlation between Test and Train devices														
	1080ti_1	1080ti_32	1080ti_64	2080ti_1	2080ti_32	2080ti_64	titan_rtx_1	titan_rtx_32	titan_rtx_64	titanx_1	titanx_32	titanx_64	titanxp_1	titanxp_32	titanxp_64
gold_6226	0.207	0.241	0.323	0.178	0.214	0.297	0.184	0.209	0.274	0.232	0.270	0.344	0.222	0.250	0.303
essential_ph_1	0.258	0.338	0.414	0.240	0.312	0.395	0.268	0.313	0.388	0.300	0.379	0.427	0.286	0.362	0.406
samsung_s7	0.307	0.372	0.421	0.293	0.349	0.406	0.312	0.352	0.404	0.347	0.402	0.435	0.337	0.388	0.416
pixel2	0.300	0.409	0.496	0.287	0.388	0.485	0.296	0.369	0.466	0.328	0.449	0.512	0.318	0.425	0.486

Table 22. FBNet task train-test correlations. Rows are test devices, Column headers are training devices.

Device	Type	NB201	FBNet
HELP & HW-NAS-Bench (Lee et al., 2021b; Li et al., 2021)			
1080ti_1	GPU	✓	✓
2080ti_1	GPU	✓	✓
1080ti_32	GPU	✓	✓
2080ti_32	GPU	✓	✓
1080ti_256	GPU	✓	✓
2080ti_256	GPU	✓	✓
titan_rtx_1	GPU	✓	✓
titanx_1	GPU	✓	✓
titanxp_1	GPU	✓	✓
titan_rtx_32	GPU	✓	✓
titanx_32	GPU	✓	✓
titanxp_32	GPU	✓	✓
titan_rtx_256	GPU	✓	✓
titanx_256	GPU	✓	✓
titanxp_256	GPU	✓	✓
gold_6240	CPU	✓	✓
silver_4114	CPU	✓	✓
silver_4210r	CPU	✓	✓
gold_6226	CPU	✓	✓
samsung_a50	mCPU	✓	✓
pixel3	mCPU	✓	✓
samsung_s7	mCPU	✓	✓
essential_ph_1	mCPU	✓	✓
pixel2	mCPU	✓	✓
fpga	FPGA	✓	✓
raspi4	eCPU	✓	✓
eyeriss	ASIC	✓	✓

Device	Type	NB201	FBNet
EAGLE(Dudziak et al., 2020)			
core_i7_7820x_fp32	CPU	✓	✗
snapdragon_675_kryo_460_int8	mCPU	✓	✗
snapdragon_855_kryo_485_int8	mCPU	✓	✗
snapdragon_450_cortex_a53_int8	mCPU	✓	✗
edge_tpu_int8	eTPU	✓	✗
gtx_1080ti_fp32	GPU	✓	✗
jetson_nano_fp16	eGPU	✓	✗
jetson_nano_fp32	eGPU	✓	✗
snapdragon_855_adreno_640_int8	mGPU	✓	✗
snapdragon_450_adreno_506_int8	mGPU	✓	✗
snapdragon_675_adreno_612_int8	mGPU	✓	✗
snapdragon_675_hexagon_685_int8	mDSP	✓	✗
snapdragon_855_hexagon_690_int8	mDSP	✓	✗

Table 23. Devices used in our paper and their categories (type). Note that we are referring to latency measurements on the same device with different batch sizes as a new device as well, this is because in some cases, there is a low correlation between these measurements.

Task Index	Type	Devices
ND	Train	1080ti_1 1080ti_32 1080ti_256 silver_4114 silver_4210r samsung_a50 pixel3 essential_ph_1 samsung_s7
	Test	titan_rtx_256 gold_6226 fpga pixel2 raspi4 eyeriss
N1	Train	embedded_tpu_edge_tpu_int8 eyeriss mobile_gpu_snapdragon_675_adreno_612_int8 mobile_gpu_snapdragon_855_adreno_640_int8 pixel3
	Test	1080ti_1 titan_rtx_32 titanxp_1 2080ti_32 titan_rtx_1
N2	Train	1080ti_1 1080ti_32 titanx_32 titanxp_1 titanxp_32
	Test	embedded_gpu_jetson_nano_fp16 embedded_tpu_edge_tpu_int8 mobile_dsp_snapdragon_675_hexagon_685_int8 mobile_dsp_snapdragon_855_hexagon_690_int8 pixel3
N3	Train	desktop_gpu_gtx_1080ti_fp32 embedded_gpu_jetson_nano_fp16 eyeriss mobile_dsp_snapdragon_675_hexagon_685_int8 mobile_gpu_snapdragon_855_adreno_640_int8
	Test	1080ti_1 2080ti_1 titanxp_1 2080ti_32 titanxp_32

Table 24. Hardware devices for NASBench-201

Task Index	Type	Devices
N4	Train	desktop_cpu_core_i7_7820x_fp32 embedded_gpu_jetson_nano_fp32 embedded_tpu_edge_tpu_int8 eyeriss mobile_cpu_snapdragon_855_kryo_485_int8 mobile_dsp_snapdragon_675_hexagon_685_int8 mobile_dsp_snapdragon_855_hexagon_690_int8 mobile_gpu_snapdragon_675_adreno_612_int8 mobile_gpu_snapdragon_855_adreno_640_int8 pixel2
	Test	1080ti_1 2080ti_1 titan_rtx_1
N2	Train	titan_rtx_1 titan_rtx_32 titanxp_1 2080ti_1 titanx_1 1080ti_1 titanx_32 titanxp_32 2080ti_32 1080ti_32 gold_6226 samsung_s7 silver_4114 gold_6240 silver_4210r samsung_a50 pixel2
	Test	eyeriss desktop_gpu_gtx_1080ti_fp32 embedded_tpu_edge_tpu_int8

Table 25. Hardware devices for NASBench-201

Table 26. Hardware devices for FBNet

Task Index	Type	Devices
FD	Train	1080ti_1 1080ti_32 1080ti_64 silver_4114 silver_4210r samsung_a50 pixel3 essential_ph_1 samsung_s7
	Test	fpga raspi4 eyeriss
F1	Train	2080ti_1 essential_ph_1 silver_4114 titan_rtx_1 titan_rtx_32
	Test	eyeriss fpga raspi4 samsung_a50 samsung_s7
F2	Train	essential_ph_1 gold_6226 gold_6240 pixel3 raspi4
	Test	1080ti_1 1080ti_32 2080ti_32 titan_rtx_1 titanxp_1
F3	Train	essential_ph_1 pixel2 pixel3 raspi4 samsung_s7
	Test	1080ti_1 1080ti_32 2080ti_1 titan_rtx_1 titan_rtx_32
F4	Train	1080ti_64 2080ti_1 eyeriss gold_6226 gold_6240 raspi4 samsung_s7 silver_4210r titan_rtx_1 titan_rtx_32
	Test	1080ti_1 pixel2 essential_ph_1
FA	Train	1080ti_1 1080ti_32 1080ti_64 2080ti_1 2080ti_32 2080ti_64 titan_rtx_1 titan_rtx_32 titan_rtx_64 titanx_1 titanx_32 titanx_64 titanxp_1 titanxp_32 titanxp_64
	Test	gold_6226 essential_ph_1 samsung_s7 pixel2