On Latency Predictors for Neural Architecture Search

Yash Akhauri [1] Mohamed S. Abdelfattah [1]

Abstract

Efficient deployment of neural networks (NN) requires the co-optimization of accuracy and latency. For example, hardware-aware neural architecture search has been used to automatically find NN architectures that satisfy a latency constraint on a specific hardware device. Central to these search algorithms is a predictive model that is designed to provide a hardware latency estimate for a candidate NN architecture. Recent research has shown that the sample efficiency of these predictive models can be greatly improved through pre-training on some *training* devices with many samples, and then transferring the predictor on the *test* (target) device. Transfer learning and meta-learning methods have been used for this, but often exhibit significant performance variability. Additionally, the evaluation of existing latency predictors has been largely done on hand-crafted training/test device sets, making it difficult to ascertain design features that compose a robust and general latency predictor. To address these issues, we introduce a comprehensive suite of latency prediction tasks obtained in a principled way through automated partitioning of hardware device sets. We then design a general latency predictor to comprehensively study (1) the predictor architecture, (2) NN sample selection methods, (3) hardware device representations, and (4) NN operation encoding schemes. Building on conclusions from our study, we present an end-to-end latency predictor training strategy that outperforms existing methods on 11 out of 12 difficult latency prediction tasks, improving latency prediction by 22.5% on average, and up to to 87.6% on the hardest tasks. By utilizing our latency predictor, we are able to speed up HW-Aware NAS by $5.8\times$ in wall-clock time. Our code is available at https://github.com/abdelfattah-lab/nasflat_latency.

1 Introduction

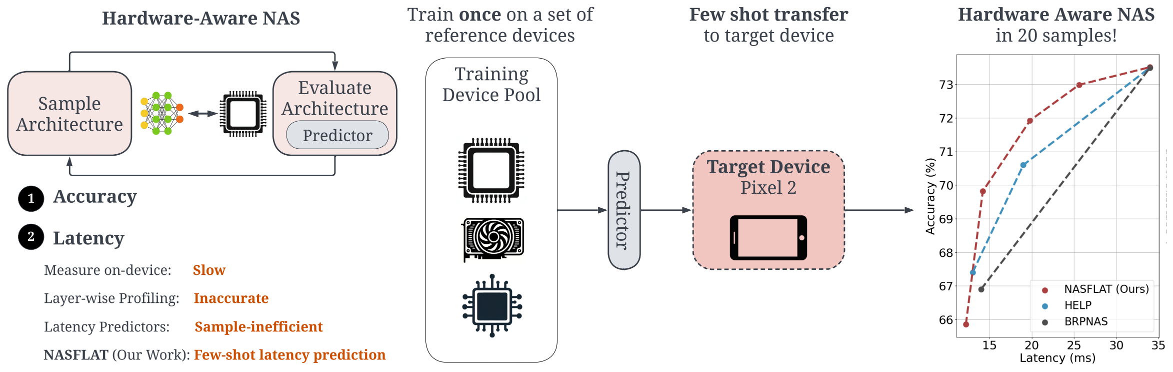
With recent advancements in deep learning (DL), neural networks (NN) have become ubiquitous, serving a wide array of tasks in different deployment scenarios. With this ubiquity, there has been a surge in the diversity of hardware devices that NNs are deployed on. This presents a unique challenge as each device has its own attributes and the same NN may exhibit vastly different latency and energy characteristics across devices. Therefore, it becomes pivotal to co-optimize both accuracy and latency to meet stringent demands of real-world deployment (Tan et al., 2019; Cai et al.; Wu et al., 2019; Xu et al., 2020; Cai et al., 2020; Wang et al., 2020). The simplest way to include latency optimization is to profile the latency of a NN on its target device. However, this becomes costly and impractical when there are multiple small changes performed on the NN architecture either manually, or automatically through neural architecture search (NAS) (Benmeziane et al., 2021). Not to mention that hardware devices are often not even available, or their software stacks do not yet support all NN architecture variants (Elsken, 2023), making it *impossible* to perform hardware-aware NN optimizations. For these reasons, much research has investigated statistical latency predictors that can accurately model hardware device latency with as few on-device measurements as possible.

Early work has used FLOPs as a proxy for latency (Yu et al., 2020), while others created layerwise latency models (Cai et al.). However, both of these approaches generally performed poorly and could not adequately represent end-to-end device latency. More recent work has used Graph Convolutional Networks (GCNs) (Dudziak et al., 2020) with much higher success in predicting the latency of NN architectures. However, a large number of NN latency samples needed to be gathered from each device for accurate modeling. To mitigate this, HELP (Lee et al., 2021b) and MultiPredict (Akhauri & Abdelfattah, 2023) leverage transfer learning to train latency predictors with only a few NN latency samples. In this paradigm, a predictor is first trained on a large set of *training* devices (the training stage), and then through few-shot meta-learning, fine-tuned to predict latency on a set of *test* devices (the transfer stage). This latter transfer stage can be performed efficiently using only a few sample measurements thus enabling the creation of latency predictors for new hardware devices in an inexpen-

[1]Cornell University. Correspondence to: Yash Akhauri <ya255@cornell.edu>, Mohamed S. Abdelfattah <mohamed@cornell.edu>.

Proceedings of the 5th MLSys Conference, Santa Clara, CA, USA, 2024. Copyright 2024 by the author(s).

Figure 1. Predictors are central to hardware-aware neural architecture search, as they enable quick evaluation of candidate architectures. Latency predictors require many training samples, but can be made more sample-efficient through pre-training on a set of training devices.


Hardware-Aware NAS
Sample Architecture
Evaluate Architecture
Predictor
1 Accuracy
2 Latency
Measure on-device: Slow
Layer-wise Profiling: Inaccurate
Latency Predictors: Sample-inefficient
NASFLAT (Our Work): Few-shot latency prediction
Train once on a set of reference devices
Training Device Pool
Predictor
Few shot transfer to target device
Target Device Pixel 2
Hardware Aware NAS in 20 samples!
Accuracy (%)
Latency (ms)
NASFLAT (Ours)
HELP
BRPNAS

sive way. Figure 1 illustrates the training and transfer of few-shot latency predictors.

This new class of few-shot latency predictors has become very practical and attractive for use within NAS and other NN latency optimization flows. However, we have identified key shortcomings of existing works, as well as open research questions relating to the design of such predictors. First, the choice of the few NN latency samples for transferring a predictor are critical. Prior work has largely chosen this handful of NN architectures randomly but this often results in very high variance in predictive ability (Lee et al., 2021b; Akhauri & Abdelfattah, 2023). Simply put, the predictor performance is directly linked to the choice of those few samples. Second, multi-hardware latency predictors require an additional input that represents the hardware device for both the pretraining and transfer phases. One-hot encoding or a vector of latency measurements were used in the past for this purpose. Third, NN operations were represented in the same way across different devices even though different operations exhibit different properties inherent to each device's hardware architecture and software compilation stack. Finally, the predictor architecture itself was largely reused from prior work (Dudziak et al., 2020) without explicit modifications for multi-device hardware predictors. To address these main points, our work makes the following contributions:

1. We investigate and empirically test different NN sampling methods for few-shot latency predictors, demonstrating a 5% improvement compared to random sampling (Lee et al., 2021b), while requiring no additional samples, unlike uniform latency sampling (Nair et al., 2022).
2. We introduce hardware-specific NN operation embeddings to modulate NN encodings based on each hardware device, demonstrating a 7.8% improvement in latency prediction. We additionally investigate the impact of supplemental encodings including unsupervised (Arch2Vec) (Yan et al., 2020), computationally aware (CATE) (Yan et al., 2021), and metric based (ZCP) (Abdelfattah et al., 2021) encodings resulting in 6.2% improvement in prediction accuracy.
3. Drawing from our evaluations on 12 experimental settings, we present **NASFLAT**, **N**eural **A**rchitecture **S**ampler And **F**ew-Shot **Lat**ency Predictor, a multi-device latency predictor architecture which combines our graph neural network with an effective sampler, supplementary encodings and transfer learning to deliver an average latency predictor performance improvement of 22.5%.

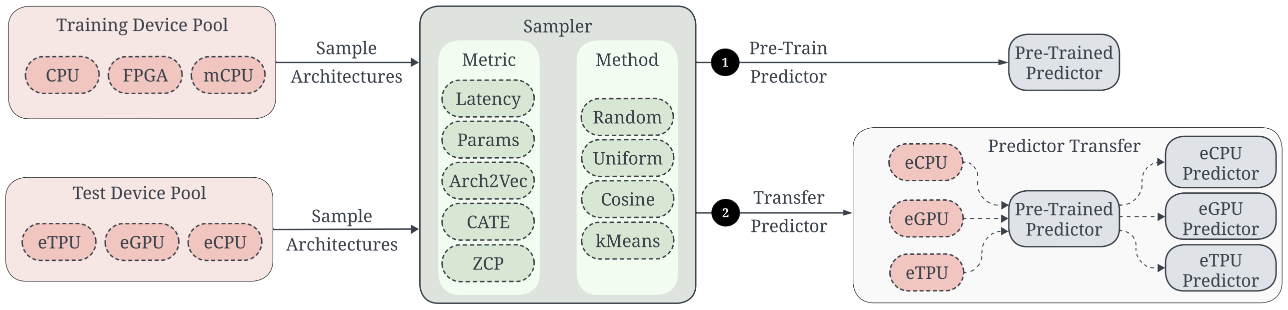
Our detailed investigation offers insights into effective few-shot latency predictor design, and results in improvements up to 87.6% on the most challenging prediction tasks (N2, FA, F2, F3 in Table 7), compared to prior work. Existing latency prediction techniques often incur higher NAS costs owing to their sample inefficiency or complicated second-order transfer strategies. When evaluating end-to-end NAS, our approach demonstrates a 5.8× speed-up in wall-clock time dedicated to latency predictor fine-tuning and prediction, compared to the best existing methods.

2 Related Work

2.1 Hardware Latency Predictors

Latency predictors enable NAS to co-optimize latency and accuracy (Tan et al., 2019; Cai et al.; Wu et al., 2019; Xu et al., 2020; Cai et al., 2020; Wang et al., 2020; Chai et al., 2023). Latency prediction methods have evolved from proxy based methods such as FLOPs (Yu et al., 2020) to learning-based methods. This is largely because such proxies often do not correlate strongly with latency at deployment. To get better estimates for latency, some works used layer-wise latency prediction methods by measuring the latency for each operation and summing up the operations that the neural network has via a look-up table (Cai et al.). However,

Figure 2. A few-shot latency predictor training pipeline. (1) We pre-train a predictor on a set of training devices. (2) The trained predictor can be adapted to any target device with just a few samples, using transfer learning. In this pipeline, the methodology used to sample neural networks as well as the predictor architecture play a key role.



this method does not account for operation pipelining or any compiler optimizations that may take place when multiple layers are executed consecutively.

BRP-NAS (Dudziak et al., 2020) takes into account such complexities by learning an end-to-end latency predictor that is trained on latency samples from the target device. However, the latency measurements of a large number of NN architectures are required to train the predictor from scratch. HELP (Lee et al., 2021b) employs a pool of reference devices to train its predictor and utilizes meta-learning techniques to adapt this predictor to a new device. The transfer of a predictor from some *source* (training) devices to a *target* (test) device significantly improves the sample efficiency of predictors. MultiPredict (Akhauri & Abdelfattah, 2023) facilitates predictor transfer across search spaces through unified encodings based on zero-cost proxies or hardware latency measurements. Furthermore, MultiPredict investigates learnable hardware embeddings to represent different hardware devices within predictors. In our work, we extend the idea of a learnable hardware embedding to make it operation specific. Having a hardware embedding that explicitly interacts with the operation embeddings of a neural network architecture can capture intricacies in compiler level optimizations when executing hardware.

When transferring a predictor from a source device to a target device, the choice of samples used for few-shot learning plays a key role in the final performance. MAPLE-Edge(Nair et al., 2022) investigates the impact of using the training device set latencies as reference for architecture to sample from the target device. For very large spaces, latencies of a sufficient diversity of neural network architectures may not be available even on the training devices. To address this, we look at methods of sampling a diverse set of neural networks from a target device which does not depend on latency or accuracy.

2.2 Encodings for NN representation

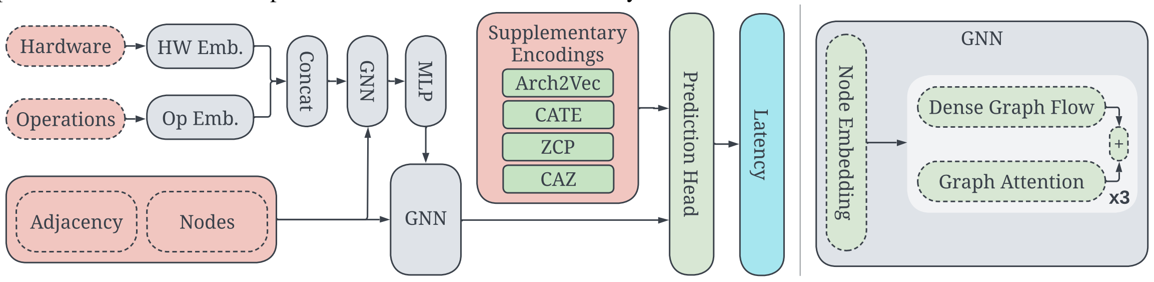
Early research in building predictors for accuracy and latency focused on using the adjacency and operation matrices to represent the directed acyclic graph (DAG) for the neural network architecture into a flattened vector to encode architectures (White et al., 2020). HELP (Lee et al., 2021b) and MultiPredict (Akhauri & Abdelfattah, 2023) use the flattened one-hot operation matrix for the FBNet space with a multi layer perceptron for the latency and accuracy predictor. BRPNAS (Dudziak et al., 2020) employed a GCN with the adjacency-operation matrices as an input to build a latency and accuracy predictor for NASBench-201. More recently, works such as MultiPredict investigated the effect of capturing broad architectural properties by generating a vector of zero-cost proxies and hardware latencies to represent NNs. Additionally, there has been significant work in the field of encoding neural networks, notably the unsupervised learned encoding introduced in Arch2Vec (Yan et al., 2020) which uses a graph auto-encoder to learn compressed latent representation for an NN architecture. Similarly, CATE (Yan et al., 2021) leveraged concepts from masked language modeling to learn encodings for computationally similar architectures with a transformer. In our work, we leverage these NN encodings to sample diverse architectures in the neural architecture search space. We also leverage these encodings to provide additional architectural information to our latency predictor as shown in Figure 2.

3 Latency Predictor Design

3.1 Predictor Architecture

The design of the predictor itself plays a key role in improving the sample efficiency of latency predictors. BRP-NAS (Dudziak et al., 2020) used a GCN predictor to capture information about both the NN operations and connectivity. The same predictor has subsequently been used by HELP (Lee et al., 2021b). TA-GATES (Ning et al., 2022) further enhanced this predictor architecture with residual

Figure 3. Our model architecture maintains separate operation and hardware embeddings which are concatenated and passed to a GNN to refine and contextualize the embeddings with respect to the overall neural architecture. This serves as the architecture embedding and is passed to another GNN along with the adjacency and node information. Optionally, supplementary encodings can be concatenated with the output of the GNN and fed to a prediction head to estimate the latency of the architecture.



connections in the GCN module and a *training-analogous* operation update methodology, by maintaining a backward graph neural network module. This allows the iterative refinement of operation embeddings to provide more information about the architecture. We investigate the impact of these and other key components of predictor design in the Appendix (A.4), and design a latency predictor that accounts for both operation and hardware embeddings. We subsequently use this predictor for all experiments in this paper.

Figure 3 shows how we maintain separate embedding tables for the hardware device and operations. From our ablations in A.4, we find that a single GNN module to refine the operation embeddings is sufficient. To capture the complex interactions such as layer pipelining and operation fusion, we further incorporate the hardware embedding into each of the operation embeddings via concatenation. This joint embedding is then passed to a small operation GNN along with the adjacency and node embedding. Then, the embedding of the output node of the GNN is provided to an MLP which provides the hardware-operation joint embedding for the second (main) GNN used for modeling an input NN. Finally, supplementary encoding (discussed more below) can be concatenated with the GNN output and provided to a prediction head that can then output a latency estimate.

3.2 GNN Module Design

Existing Graph Convolutional Networks (GCNs) experience an over-smoothing challenge, leading to a loss of discriminative information in the node embedding as aggregation layers increase. Addressing this, GATES (Ning et al., 2023) introduced the Dense Graph Flow (DGF) module, which employs residual connections to maintain discriminative features across nodes. Furthermore, the Graph Attention (GAT) methodology, distinct from DGF, incorporates an attention mechanism during node aggregation. GAT evaluates node interactions through an attention layer. An operation attention mechanism, along with LayerNorm, refines information aggregation and ensures stable training, respectively. Further details of their implementation are in A.3.1. In our latency predictor, we use an ensemble of DGF and GAT modules.

3.3 Supplementary Encodings

Supplementary encodings are different ways to represent the input NN, and therefore may help contextualize the relations between a NN with respect to the entire search space. This can be useful for few-shot transfer of latency predictors. Learned encodings like Arch2Vec (Yan et al., 2020), ZCP (Akhauri & Abdelfattah, 2023) and CATE (Yan et al., 2021) provide distinct representations that allows them to effectively distinguish between various neural network (NN) architectures. For example, CATE (Yan et al., 2021) captures computational characteristics of NNs through its latent representations formed by computational clustering. Simultaneously, ZCP offers insights at the architectural level, acting as proxies that might correlate with accuracy.

To enhance the robustness and accuracy of our latency predictor, we integrate these encodings into its structure. Specifically, Arch2Vec, CATE, and ZCP encodings are introduced as supplementary inputs to the predictor head of our latency predictor. As illustrated in Figure 3, these encodings are fed into the MLP prediction head subsequent to the node aggregation phase. This augmentation not only incorporates the rich structural and computational characteristics of NN architectures but also helps the predictor to make better-informed latency estimates. In our experiments, we observed that incorporating architectural-level information from encodings can boost the sample-efficiency of our latency predictors. We also introduce the CAZ encoding, a combined representation formed by concatenating CATE, Arch2Vec, and ZCP, aiming to leverage the combined strengths of all three representations.

3.4 Transfer Of Pre-Trained Predictor

To pretrain the predictor, we form a large dataset from a number of source devices. This conventional training step is the same as prior work (Lee et al., 2021b; Akhauri & Abdelfattah, 2023). Once this pre-training phase is concluded, the subsequent step is the adaptation to a target hardware device. In alignment with the methodology described in MultiPredict (Akhauri & Abdelfattah, 2023), the predictor undergoes a fine-tuning process using the samples from the target device. The learning rate is re-initialized and fine-tuning of the predictor is conducted on the target device. We find that this is sufficient to calibrate the predictor to offer accurate latency estimates on the unseen device.

4 Neural Network Samplers

One of the key aspects of latency predictor training, especially in the low-sample count regime is choosing diverse neural networks. If all the samples profiled on the target device have similar computational characteristics, the predictor may not gather enough information to generalize. As depicted in Figure 2, one of the key aspects of predictor training and transfer is the sampler, which needs to select a diverse set of neural architectures to benchmark for few-shot learning. MAPLE-Edge (Nair et al., 2022) uses latencies on a set of reference devices to identify architectures with distinct computational properties. While this is a very effective strategy to identify computationally distinct neural networks, this requires a very large number of on-device latency measurements—a key parameter that we would like to minimize.

In this section, we investigate methods of encoding NN architectures. We look at the impact of these encodings in helping us sample more diverse architectures. One key benefit of using these encodings to sample diverse architectures is that we no longer depend on reference device latency measurements in the sampling process, thus improving overall predictor training efficiency.

4.1 Neural Network Encodings

A foundational aspect of hardware-aware NAS is to optimize an objective function $\ell : A \to \mathbb{R}$, where ℓ can quantify several performance metrics such as accuracy, latency, energy (Dudziak et al., 2020). A represents the NN search space, and architectures $a \in A$ can be represented as adjacency - operation matrices (White et al., 2020). There are several methods that introduce alternative methods to represent a (Yan et al., 2020; 2021; Akhauri & Abdelfattah, 2023). In this section, we look at some of these methods of encoding neural networks.

Learned encodings aim to represent the structural properties of a neural architecture in a latent vector without utilizing accuracy. Arch2Vec (Yan et al., 2020) uses a variational graph isomorphism autoencoder to learn to regenerate the adjacency-operation matrix. CATE (Yan et al., 2021) introduces a transformer that uses computationally similar architecture pairs (clustered by similar FLOPs or parameter count) to learn encodings. Naturally, the CATE encoding clusters architectures that have similar computational properties.

Zero-Cost Proxies (ZCP) encode neural networks as a vector of metrics. Each of these metrics attempt to encode properties of a neural network that may correlate with accuracy. Distinct connectivity patterns and operation choices (via varying adjacency-operation matrices) would initialize NNs that exhibit varied accuracy and latency characteristics. Thus, ZCP can implicitly capture architectural properties of neural networks, but does not contain explicit structural information.

4.2 Encoding-based Samplers

Encodings like ZCP, Arch2Vec, and CATE condense a broad spectrum of architectural information into a latent space. While Arch2Vec compresses the adjacency-operation matrix, capturing its intrinsic structure, CATE identifies and groups computationally similar architectures. In contrast, ZCPs capture global properties of the neural network that may correlate with accuracy or may encode operator level information about the architecture. Such encodings, collectively contain a rich representation of the entire neural architecture design space which can be used to decide which architectures to obtain the latency for on the target device.

For the ZCP encoding, we use 13 zero cost proxies, and generate 32-dimensional vectors for the Arch2Vec and CATE encodings. We further introduce an encoding "CAZ", which combines the CATE, Arch2Vec and ZCP. Given the richness and diversity of the encoded representation of neural architectures, a systematic approach to selection becomes essential. We thus investigate two methodologies of using the encoding to identify architectures to sample for few-shot transfer to a target device.

Cosine Similarity and KMeans Clustering: Through our framework, we leverage cosine similarity (Lahitani et al., 2016)—an intuitive metric of vector similarity—to discern architectures that may have distinct properties. Focusing on structures with reduced average cosine similarities ensures a wider design space coverage, potentially identifying 'outlier' architectures. Concurrently, utilizing the KMeans clustering algorithm (MacQueen et al., 1967), we categorize encoded vectors into distinct groups, and opt for the one closest to the centroid of each cluster. The rationale is that these architectures are most representative of their respective clusters and hence provide a good spread across the design space.

Table 1. The impact of operation-wise hardware embedding (OPHW) and hardware embedding initialization (INIT) on latency predictor performance. Both optimizations consistently demonstrate an improvement.

OPHW	ND	N1	N2	N3	N4	NA	FD	F1	F2	F3	F4	FA
✗	$0.804_{0.026}$	$0.701_{0.056}$	$0.763_{0.044}$	$0.680_{0.042}$	$\mathbf{0.757_{0.061}}$	$0.660_{0.030}$	$0.839_{0.010}$	$0.753_{0.031}$	$0.745_{0.081}$	$0.685_{0.085}$	$0.813_{0.030}$	$0.566_{0.081}$
✓	$\mathbf{0.806_{0.038}}$	$\mathbf{0.719_{0.050}}$	$\mathbf{0.795_{0.034}}$	$\mathbf{0.704_{0.058}}$	$0.753_{0.052}$	$\mathbf{0.664_{0.059}}$	$\mathbf{0.845_{0.009}}$	$\mathbf{0.757_{0.048}}$	$\mathbf{0.769_{0.076}}$	$\mathbf{0.694_{0.076}}$	$\mathbf{0.832_{0.026}}$	$\mathbf{0.628_{0.103}}$
INIT	ND	N1	N2	N3	N4	NA	FD	F1	F2	F3	F4	FA
✗	$0.794_{0.038}$	$0.701_{0.046}$	$0.791_{0.036}$	$\mathbf{0.718_{0.055}}$	$0.750_{0.050}$	$0.658_{0.080}$	$0.707_{0.158}$	$0.609_{0.119}$	$0.754_{0.064}$	$0.655_{0.084}$	$0.659_{0.158}$	$0.559_{0.118}$
✓	$\mathbf{0.806_{0.038}}$	$\mathbf{0.719_{0.050}}$	$\mathbf{0.795_{0.034}}$	$0.704_{0.058}$	$\mathbf{0.753_{0.052}}$	$\mathbf{0.664_{0.059}}$	$\mathbf{0.845_{0.009}}$	$\mathbf{0.757_{0.048}}$	$\mathbf{0.769_{0.076}}$	$\mathbf{0.694_{0.076}}$	$\mathbf{0.832_{0.026}}$	$\mathbf{0.628_{0.103}}$

Table 2. Device sets for NASBench-201 and FBNet. S, T indicate source and target device pools respectively. Each *device pool* may contain more than one device, full details in the Appendix. Unless otherwise specified, we report the *average* correlation and standard deviation across trials *and* target (T) devices.

Devices	NB201						FBNet					
	ND	N1	N2	N3	N4	NA	FD	F1	F2	F3	F4	FA
DSP	-	-	T	-	-	-	-	-	-	-	-	-
CPU	ST	-	-	S	S	ST	ST	S	S	-	S	T
GPU	ST	T	S	T	T	S	ST	S	T	T	ST	S
FPGA	T	-	-	-	-	-	-	T	S	S	-	-
ASIC	T	S	-	S	S	T	-	T	-	-	S	-
eTPU	-	S	T	-	S	T	-	-	-	-	-	-
eGPU	-	-	T	S	S	-	-	-	-	-	-	-
eCPU	T	-	T	-	-	-	S	T	-	-	S	-
mGPU	-	S	-	S	S	-	-	-	-	-	-	-
mCPU	S	S	-	-	S	S	-	S	S	S	ST	T
mDSP	-	-	-	S	S	-	-	-	-	-	-	-

Algorithm 1 Methodology to partition device sets.

```
Input: Graph $G$ (negative correlations), integers $m$, $n$
Output: Modified graph $B$
$b\_m, b\_n =$ kernighan-lin(G)
{Initialize bipartite graph with correlations}
$B =$ initBipartite($b\_m, b\_n$, correlations)
while len(B[0]) $\neq$ m or len(B[1]) $\neq$ n do
  {Identify disjoint device sets $U$ and $V$}
  l, r = B[0], B[1]
  {Remove node with highest correlation.}
  if len(B[0]) $>$ m then
    removeMaxWeightNode(B, l)
  end if
  if len(B[1]) $>$ n then
    removeMaxWeightNode(B, r)
  end if
end while
```

5 Hardware Embeddings

HELP and MultiPredict (Lee et al., 2021b; Akhauri & Abdelfattah, 2023) investigate different methods of representing hardware, as an assigned device index, a vector of architectural latency measurements, or as a learnable hardware embedding table, which is relayed to the predictor for identifying devices. However, such an approach potentially oversimplifies the intricate dynamics of neural network deployment on hardware, thereby introducing the need for an interaction between the operation and hardware embedding. In this section, we discuss a methodology to initialize and utilize hardware embeddings to better model the dynamics of the target hardware.

5.1 Operation Specific Hardware Embedding

From a hardware perspective, the location of an operation in relation to its preceding and succeeding layers can considerably influence overall latency. This can be attributed to optimizations such as layer pipelining, where operations are organized in a staggered manner to maximize hardware utilization. Additionally, optimizations such as layer fusion which combine adjacent layers to streamline computations further underscore the importance of operation placement.

The latency predictor accepts the adjacency matrix, node, and operation indices as its inputs. The operation index is utilized to retrieve a learnable operation embedding from an embedding table, which encodes the properties and behaviors of the respective operation, however, when modeling latencies for various hardware devices, this singular operation embedding might not fully capture the nuances of each hardware. As depicted in Figure 3, we concurrently incorporate hardware-specific embeddings into our predictor, such that we are able to model the interaction between an operation and the hardware depending on its nature and position. Thus, the operation-specific hardware embedding introduces a more granular approach wherein each operation within the neural network is concatenated with a specific hardware embedding from an embedding table. Such a strategy not only encapsulates the intrinsic characteristics of the operation but also embeds information regarding how that specific operation would behave on the given hardware.

5.2 Hardware Embedding Initialization

When adding a new target device, a good initialization for its hardware embedding is critical. For this, we gauge the computational correlation of the target device latency with each of the training devices. By identifying the training device with the highest correlation, we can use its learned

Table 3. On 10 out of 12 device sets, there is a benefit in using learned or zero-cost encodings for training-transfer sample selection.

Sampler	ND	N1	N2	N3	N4	NA	Geometric Mean
Latency (Oracle)	$0.929_{0.027}$	$0.960_{0.011}$	$0.793_{0.078}$	$0.951_{0.021}$	$0.919_{0.055}$	$0.851_{0.039}$	0.898
Random	$0.911_{0.038}$	$0.946_{0.026}$	$0.757_{0.052}$	$0.934_{0.032}$	$0.940_{0.026}$	$0.790_{0.070}$	**0.876**
Params	$0.898_{0.068}$	$0.934_{0.038}$	**$0.767_{0.068}$**	$0.918_{0.033}$	$0.940_{0.032}$	$0.801_{0.095}$	0.873
Arch2Vec	$0.912_{0.045}$	$0.931_{0.046}$	$0.741_{0.073}$	$0.930_{0.036}$	$0.907_{0.069}$	**$0.849_{0.035}$**	0.875
CATE	$0.893_{0.045}$	$0.937_{0.036}$	$0.761_{0.090}$	$0.935_{0.032}$	**$0.945_{0.038}$**	$0.767_{0.136}$	0.869
ZCP	$0.883_{0.075}$	$0.956_{0.039}$	$0.636_{0.170}$	$0.924_{0.051}$	$0.883_{0.071}$	$0.729_{0.212}$	0.826
CAZ	**$0.925_{0.046}$**	**$0.957_{0.028}$**	$0.761_{0.107}$	**$0.935_{0.025}$**	$0.866_{0.091}$	$0.680_{0.336}$	0.847

Sampler	FD	F1	F2	F3	F4	FA	Geometric Mean
Latency (Oracle)	$0.755_{0.048}$	$0.707_{0.052}$	$0.832_{0.035}$	$0.849_{0.022}$	$0.699_{0.077}$	$0.624_{0.112}$	0.740
Random	$0.665_{0.187}$	$0.642_{0.121}$	$0.801_{0.063}$	**$0.809_{0.050}$**	$0.658_{0.113}$	$0.615_{0.115}$	0.694
Params	$0.735_{0.073}$	$0.689_{0.070}$	$0.794_{0.078}$	$0.791_{0.062}$	$0.604_{0.239}$	$0.551_{0.146}$	0.687
Arch2Vec	**$0.754_{0.071}$**	**$0.699_{0.046}$**	$0.790_{0.065}$	$0.782_{0.083}$	**$0.739_{0.054}$**	**$0.631_{0.169}$**	**0.730**
CATE	$0.663_{0.132}$	$0.692_{0.079}$	$0.778_{0.078}$	$0.780_{0.076}$	$0.645_{0.118}$	$0.552_{0.147}$	0.680
ZCP	$0.744_{0.060}$	$0.665_{0.111}$	$0.789_{0.069}$	$0.801_{0.055}$	$0.734_{0.051}$	$0.586_{0.161}$	0.715
CAZ	$0.696_{0.107}$	$0.635_{0.105}$	**$0.808_{0.040}$**	$0.732_{0.097}$	$0.626_{0.127}$	$0.557_{0.141}$	0.670

hardware embedding as the starting point for the target device. This method harnesses latency similarities between devices, providing a good initialization for predictions on the target device. In addition, it avoids a *cold start* when using the predictor for a new device, allowing the predictor to be functional with just a small number of latency samples on the new device.

6 EXPERIMENTS

In this section, we systematically investigate key design considerations for predictor design. We begin by looking at the effectiveness of our proposed operation specific hardware embedding as well as hardware embedding initialization. We further investigate the impact that graph neural network module design has on predictor efficacy, looking at graph convolutional networks, graph attention networks and their ensemble. We then study the impact of neural network encoding strategies on selection of neural network architectures for transferring predictors to a target device. By supplementing our predictor with additional NN encodings, we provide additional information coveying the relative performance of NNs within a search space—our ablation shows improved prediction. Finally, we combine our empirically-driven optimizations on sampling, encodings, hardware embeddings, and predictor architecture to design a state-of-the-art latency predictor. Our evaluation encompasses both the Spearman Rank Correlation coefficient of predicted latency and ground truth on multiple device sets, in addition to end-to-end HW-Aware NAS.

6.1 Designing Evaluation Sets

One of the key challenges in evaluating HW-Aware NAS is the lack of standardized evaluation sets and hardware latency data-sets. HW-NAS-Bench (Li et al., 2021), HELP (Lee et al., 2021b) and EAGLE (Dudziak et al., 2020) collectively open-source latencies for a wide range of hardware on the NASBench-201 and FBNet design spaces, making HW-Aware NAS evaluation easier. However, current device sets showed high training-test correlation (Akhauri & Abdelfattah, 2023). For NASBench-201 and FBNet, high training-test correlation device sets are denoted as 'ND' and 'FD', respectively. A key shortcoming in these existing works is that the devices that the predictor is evaluated on is hand-picked, exhibiting a high correlation between the training devices and test devices, a property that is not guaranteed in practice. To circumvent this limitation, we employ an automated algorithmic strategy to design training and test devices to maintain low mutual correlation. Initially, we compute the Spearman correlation coefficient of latencies between all available devices to construct a graph wherein correlations between devices are used as edge weights. As Algorithm 1 shows, we then leverage the Kernighan-Lin (Kernighan & Lin, 1970) bisection method to bisect the graph, aiming to group devices with minimal intra-group correlation. We iteratively trim the bipartite graph to maintain a specified number of devices in each set. By algorithmically partitioning device sets, we ensure an objective and unbiased selection process. With this strategy, we introduce four device sets for NASBench-201 and FBNet, identified by (N1, N2, N3, N4) and (F1, F2, F3, F4) in Table 2. We also show "NA" and "FA" introduced by MultiPredict in the

Table 4. Over three device sets on two NAS spaces, there is a benefit in using supplementary encodings for representing neural networks.

Encoding	ND	N1	N2	N3	N4	NA	Geometric Mean
AdjOp	$0.959_{0.018}$	$0.971_{0.010}$	$0.848_{0.063}$	$0.966_{0.006}$	$0.965_{0.012}$	$0.893_{0.029}$	0.932
(+ Arch2Vec)	$\mathbf{0.961_{0.010}}$	$\mathbf{0.972_{0.007}}$	$0.816_{0.065}$	$\mathbf{0.968_{0.005}}$	$0.965_{0.009}$	$0.895_{0.035}$	0.927
(+ CATE)	$0.954_{0.022}$	$0.962_{0.029}$	$0.833_{0.072}$	$0.967_{0.007}$	$\mathbf{0.967_{0.008}}$	$\mathbf{0.907_{0.024}}$	0.930
(+ ZCP)	$0.955_{0.024}$	$0.968_{0.014}$	$0.855_{0.026}$	$0.963_{0.009}$	$0.962_{0.011}$	$0.896_{0.018}$	0.932
(+ CAZ)	$0.960_{0.014}$	$0.972_{0.005}$	$\mathbf{0.861_{0.036}}$	$0.965_{0.005}$	$0.966_{0.006}$	$0.899_{0.012}$	**0.936**
Encoding	FD	F1	F2	F3	F4	FA	Geometric Mean
AdjOp	$0.783_{0.035}$	$0.744_{0.032}$	$0.855_{0.021}$	$0.850_{0.026}$	$0.750_{0.066}$	$0.694_{0.060}$	0.777
(+ Arch2Vec)	$0.881_{0.016}$	$0.788_{0.027}$	$0.878_{0.020}$	$0.890_{0.012}$	$\mathbf{0.848_{0.020}}$	$0.723_{0.034}$	0.832
(+ CATE)	$0.837_{0.031}$	$0.744_{0.037}$	$0.839_{0.022}$	$0.845_{0.037}$	$0.805_{0.023}$	$0.678_{0.055}$	0.788
(+ ZCP)	$\mathbf{0.960_{0.008}}$	$\mathbf{0.842_{0.020}}$	$\mathbf{0.895_{0.018}}$	$\mathbf{0.899_{0.019}}$	$0.843_{0.028}$	$\mathbf{0.776_{0.038}}$	**0.867**
(+ CAZ)	$0.899_{0.021}$	$0.783_{0.033}$	$0.842_{0.036}$	$0.852_{0.029}$	$0.761_{0.056}$	$0.683_{0.077}$	0.800

Table 5. On NB201, GAT outperforms DGF. The difference is less evident on FBNet, we thus use an ensemble of DGF and GAT.

GNN Module	ND	N1	N2	N3
DGF	$0.814_{0.026}$	$0.741_{0.032}$	$\mathbf{0.802_{0.021}}$	$0.710_{0.042}$
GAT	$\mathbf{0.965_{0.005}}$	$\mathbf{0.848_{0.038}}$	$0.612_{0.032}$	$0.762_{0.039}$
Ensemble	$0.965_{0.011}$	$0.829_{0.046}$	$0.634_{0.029}$	$\mathbf{0.789_{0.054}}$
GNN Module	FD	F1	F2	F3
DGF	$\mathbf{0.844_{0.004}}$	$0.740_{0.055}$	$0.752_{0.076}$	$\mathbf{0.621_{0.120}}$
GAT	$0.823_{0.019}$	$\mathbf{0.749_{0.042}}$	$0.700_{0.114}$	$0.589_{0.083}$
Ensemble	$0.835_{0.007}$	$0.733_{0.059}$	$\mathbf{0.766_{0.013}}$	$0.609_{0.106}$

same table (*A* signifying the adversarial nature of the device set) in Table 2.

6.2 Experimental Setup

In each of our experiments, we pretrain our predictor on many samples from multiple source devices, then we fine-tune the predictor on only a few samples from the target devices as defined by our evaluation sets. We report the Spearman Rank Correlation Coefficient of predicted latency relative to ground-truth latency values as a measure of predictive performance. In Table 5, we investigate the GAT and DGF GNN module. In all our experiments, we decide to use the DGF-GAT ensemble as the GNN module. The appendix (Table 20) details the precise experimental settings for each of our results.

6.3 Hardware-aware Operation Embeddings

Instead of representing operations with a fixed embedding, we modulate embeddings based on each hardware device as described in Section 5. Table 1 evaluates the impact of this optimization on the predictive ability using our latency predictor. We find that on 11 out of 12 device pools, there is a positive impact of introducing the operation-wise hardware embedding. Additionally, we evaluate the impact of initializing the hardware embedding of the target device with the embedding of the most closely-correlated source device. Our results show consistent improvement from this initialization scheme as shown in Table 1. These two optimizations have empirically demonstrated their efficacy in utilizing hardware-specific operation embeddings, and mitigating new device cold start in our predictor.

6.4 Encoding-based Samplers

Here, we evaluate different sampling methods, specifically those based on uniformly sampling NNs based on different encodings. Our findings in Table 3 present a somewhat varied pattern. While encodings-based samplers were advantageous in 10 out of 12 device pools, determining the optimal sampler became a complex task dependent on the each device pool. Notably, CATE's performance was subpar, especially on FBNet. A potential reason for this could be the disparity between the FBNet search space, which possesses 9^{22} unique architectures, and the latency dataset available in HWNASBench, restricted to only 5000 architectures. Consequently, when training the CATE encoding on this limited set, computationally similar architectures do not carry much meaning with respect to the entire FBNet search space. Arch2Vec is trained similarly, but since computationally similar architectures are not required, it is able to learn a representation on a smaller space.

Delving deeper into the selection strategy used for clustering the encodings, we observed a pronounced inclination towards the cosine similarity approach. As detailed in the appendix (Table 9), Cosine similarity consistently demonstrated superior performance over KMeans. Additionally,

Table 6. We study the impact of different design choices on the performance of predictors. Each row adds a feature and also inherits the design choices above it.

	F1	F2	F3	F4	N1	N2	N3	N4
Baseline Predictor	$0.603_{0.104}$	$0.800_{0.050}$	$0.792_{0.058}$	$0.502_{0.142}$	$0.938_{0.034}$	$0.781_{0.084}$	$0.907_{0.019}$	$0.922_{0.021}$
(+ HWInit)	$0.573_{0.114}$	$0.770_{0.072}$	$0.822_{0.053}$	$0.566_{0.127}$	$0.938_{0.031}$	$0.776_{0.039}$	$0.887_{0.087}$	$0.898_{0.046}$
(+ Op$_{HW}$)	$0.610_{0.076}$	$0.801_{0.045}$	$\mathbf{0.824_{0.025}}$	$0.549_{0.078}$	$0.949_{0.017}$	$\mathbf{0.821_{0.045}}$	$\mathbf{0.938_{0.021}}$	$0.955_{0.015}$
(+ Sampler)	$0.639_{0.069}$	$0.813_{0.042}$	$0.810_{0.053}$	$0.616_{0.116}$	$\mathbf{0.962_{0.012}}$	$0.803_{0.067}$	$0.920_{0.034}$	$\mathbf{0.961_{0.007}}$
(+ Supp. Encoding)	$\mathbf{0.727_{0.048}}$	$\mathbf{0.844_{0.031}}$	$0.816_{0.057}$	$\mathbf{0.796_{0.045}}$	$0.936_{0.033}$	$0.812_{0.059}$	$0.930_{0.028}$	$0.940_{0.017}$

Table 7. We train our predictor with our proposed sampler, GAT+GCN ensemble architecture, operation-wise hardware embedding, hardware embedding initialization, and report our end-to-end predictor transfer result (Spearman Rank Correaltion). On 11 out of 12 tasks, our predictor works best for NASBench-201 and FBNet respectively. Standard deviations are reported for results we produce in our paper.

	Source	Target	ND	NA	N1	N2	N3	N4	GM
	Samples								
HELP	900	20	$0.948_{0.006}$	$0.410_{0.037}$	$0.604_{0.044}$	$0.509_{0.007}$	$0.729_{0.027}$	$0.746_{0.042}$	0.634
MultiPredict	900	20	$0.930_{0.012}$	$0.820_{0.019}$	$0.907_{0.003}$	$0.757_{0.045}$	$0.947_{0.012}$	$0.952_{0.011}$	0.882
NASFLAT	25	20	$\mathbf{0.959_{0.007}}$	$\mathbf{0.893_{0.036}}$	$\mathbf{0.967_{0.007}}$	$\mathbf{0.857_{0.029}}$	$\mathbf{0.962_{0.008}}$	$\mathbf{0.959_{0.012}}$	**0.931**
			FD	FA	F1	F2	F3	F4	GM
HELP	4000	20	0.910	0.37	$0.793_{0.028}$	$0.543_{0.036}$	$0.413_{0.015}$	$\mathbf{0.799_{0.004}}$	0.602
MultiPredict	4000	20	0.960	0.45	$0.756_{0.026}$	$0.567_{0.075}$	$0.434_{0.040}$	$0.763_{0.011}$	0.627
NASFLAT	800	20	$\mathbf{0.961_{0.007}}$	$\mathbf{0.577_{0.079}}$	$\mathbf{0.809_{0.019}}$	$\mathbf{0.871_{0.024}}$	$\mathbf{0.814_{0.046}}$	$0.734_{0.142}$	**0.784**

KMeans was occasionally unable to segment the space adequately to yield architectures (as evidenced by NaN entries).

6.5 Supplementary NN Encodings

By using supplementary NN encodings in our predictor (as shown in Figure 3), we can better represent the relative performance of NNs, especially when only a few samples are used for predictor training. From Table 4, we see that the impact of these supplementary encodings revealed an almost universal benefit: 11 out of 12 device pools displayed improved performance. This is likely due to the fact that these encodings contextualize the few target samples with respect to the broader search space more effectively.

6.6 Impact of Pre-Training Samples

Focusing on the pre-training phase, we delve into how varying sample sizes from source devices influence the Spearman rank correlation. Notably, the end-to-end performance does not consistently improve with an increasing number of samples; quite the opposite, it occasionally diminishes. This counter-intuitive phenomenon can be ascribed to a situation where the model, encountering a multitude of source devices that share high correlation, ends up overfitting to the specifics of this source device set. For instance, in "Task N4", where the training set already has a relatively low average correlation between devices, degradation of predictive performance with more samples isn't observed. However, in "Task N2", which comprises solely of GPUs, this tendency becomes more clear. These findings indicate that the diversity in the training pool is important to benefit from larger source sample training sets. Merely increasing the number of samples without assuring their diversity can hinder predictor pre-training. We perform an ablation in the appendix (Figure 7) to select a reasonable number of pre-training samples for use with our predictor in subsequent experiments.

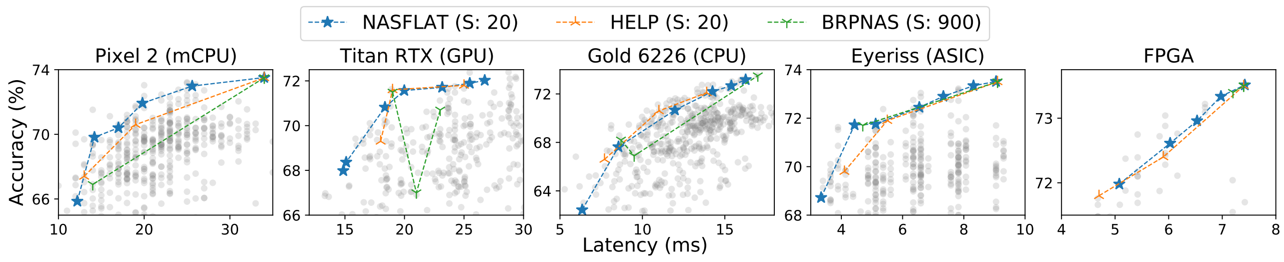
6.7 Combining our Optimizations and a Comparison to Prior Work

Table 6 lists the effect of combining all of our optimizations including the hardware-aware operation embeddings, embedding initialization, encoding-based samplers, and supplementary encodings, all performed on our predictor architecture. We call our final predictor **NASFLAT**: **N**eural **A**rchitecture **S**ampler And **F**ew-Shot **Lat**ency Predictor. On average, our first 3 optimizations bring marked improvements to the predictor performance. We also found that using our encoding-based samplers generally reduced variance, making predictor construction more reliable. This is further quantified in the appendix.

Finally, Table 7 incorporates all our optimizations to deliver state-of-the-art end-to-end latency predictor performance on 11 out of 12 device sets. We show that, especially on challenging tasks, our optimizations improve predictor accuracy

Table 8. Performance comparison of different latency estimators combined with MetaD2A for latency-constrained NAS, on CIFAR-100 dataset with NAS-Bench-201 search space. For the building time and the total NAS cost of MetaD2A+HELP, we report only time and cost during the meta-test time. The meta-training time of HELP is 25 hours, NASFLAT is 25 minutes and the time to meta-train the MetaD2A is 46 GPU hours, which is conducted only once across all unseen devices. We report averages over 10 trials. S indicates number of new samples on target device.

Device	Model	Const (ms)	Latency (ms)	Accuracy (%)	Latency Model Sample	Latency Model Building Time	Total NAS Cost (Wall Clock)	Speed Up
Unseen Device Google Pixel2	MetaD2A + BRP-NAS (Dudziak et al., 2020)	14	14	66.9	900	1120s	1220s	0.1×
	MetaD2A + HELP (Lee et al., 2021b)		13	67.4	20	25s	125s	1×
	MetaD2A + NASFLAT		$\mathbf{14.4_{3.41}}$	$\mathbf{68.53_{3.04}}$	**20**	**25s**	**29.1s**	**4.3**×
	MetaD2A + BRP-NAS (Dudziak et al., 2020)	22	34	73.5	900	1120s	1220s	0.1×
	MetaD2A + HELP (Lee et al., 2021b)		19	70.6	20	25s	125s	1×
	MetaD2A + NASFLAT		$\mathbf{22.2_{6.46}}$	$\mathbf{72.08_{0.91}}$	**20**	**25s**	**29.1s**	**4.3**×
	MetaD2A + BRP-NAS (Dudziak et al., 2020)	34	34	73.5	900	1120s	1220s	0.1×
	MetaD2A + HELP (Lee et al., 2021b)		34	73.5	20	25s	125s	1×
	MetaD2A + NASFLAT		$\mathbf{34_{0.0}}$	$\mathbf{73.5_{0.0}}$	**20**	**25s**	**29.1s**	**4.3**×
Unseen Device Titan RTX (Batch Size 256)	MetaD2A + Layer-wise Pred.	18	37	73.2	900	998s	1098s	0.1×
	MetaD2A + BRP-NAS (Dudziak et al., 2020)		21	67.0	900	940s	1040s	0.1×
	MetaD2A + HELP (Lee et al., 2021b)		18	69.3	20	11s	111s	1 ×
	MetaD2A + NASFLAT		$\mathbf{17.10_{2.65}}$	$\mathbf{69.92_{2.35}}$	**20**	**11s**	**15.4s**	**7.2**×
	MetaD2A + Layer-wise Pred.	21	41	73.5	900	998s	1098s	0.1×
	MetaD2A + BRP-NAS (Dudziak et al., 2020)		19	71.5	900	940s	1040s	0.1×
	MetaD2A + HELP (Lee et al., 2021b)		19	71.6	20	11s	111s	1 ×
	MetaD2A + NASFLAT		$\mathbf{20.47_{2.46}}$	$\mathbf{71.45_{0.45}}$	**20**	**11s**	**15.4s**	**7.2**×
	MetaD2A + Layer-wise Pred.	25	41	73.2	900	998s	1098s	0.1×
	MetaD2A + BRP-NAS (Dudziak et al., 2020)		23	70.7	900	940s	1040s	0.1×
	MetaD2A + HELP (Lee et al., 2021b)		25	71.8	20	11s	111s	1 ×
	MetaD2A + NASFLAT		$\mathbf{26.61_{4.84}}$	$\mathbf{71.9_{0.87}}$	**20**	**11s**	**15.4s**	**7.2**×



by 22.5% on average, and up to 87.6% for the hardest (F3) task when compared to HELP (Lee et al., 2021b).

6.8 Neural Architecture Search

To assess the performance on end-to-end NAS, our predictor is used within the hardware-aware NAS system presented in HELP (Lee et al., 2021b). We use the same NAS system with Metad2a (Lee et al., 2021a) as the NN search algorithm for accuracy, and our predictor to find latency. We evaluate our results using multiple metrics. Primarily, we consider the number of architecture-latency pair samples taken from the target device. These samples are crucial for constructing the latency predictor, which encompasses the total time span of sample acquisition, architecture compilation on the device and the latency measurement process. While this metric remains the same between our method and HELP, we also evaluated the total NAS cost, a combination of the time taken for few-shot transfer of predictor to the NAS experiment device, and the time spent on latency prediction during NAS. The results in Table 8 (and companion plots) demonstrate a consistent improvement in the discovered NNs, with the latency-accuracy Pareto curve dominating that of prior works in most cases. Additionally, our predictor is faster to invoke when compared to HELP, making fast NAS methods—such as Metad2a—around 5× faster.

7 Conclusion

In this paper, we systematically examined several design considerations inherent to hardware latency predictor design. We studied the influence of operation-specific hardware embeddings, graph neural network module designs, and the role of neural network encodings in enhancing predictor accuracy. Our exhaustive empirical evaluations across multiple device sets yielded key insights, such as the importance of sample diversity during pre-training and the significant impact of supplemental encodings based on their correlation with test devices. By adopting an algorithmic partitioning strategy for device set selection, we achieved a less biased evaluation of latency predictor performance. Leveraging these insights, we developed and validated NASFLAT, a latency predictor that outperformed existing works in 11 of 12 device sets. Future endeavors to refine predictors could involve exploring sophisticated transfer learning techniques akin to HELP (Lee et al., 2021b), deepening our understanding of neural network encodings' mechanisms as samplers, and investigating different sampling methodologies. Our work paves the way for more reliable and efficient use of predictors, both within NAS, and more generally in the optimization of NN architectures.

References

Abdelfattah, M. S., Mehrotra, A., Dudziak, Ł., and Lane, N. D. Zero-cost proxies for lightweight nas. *arXiv preprint arXiv:2101.08134*, 2021.

Akhauri, Y. and Abdelfattah, M. S. Multi-predict: Few shot predictors for efficient neural architecture search, 2023.

Benmeziane, H., Maghraoui, K. E., Ouarnoughi, H., Niar, S., Wistuba, M., and Wang, N. A comprehensive survey on hardware-aware neural architecture search, 2021.

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

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/pdf/1908.09791.pdf.

Chai, Y., Tripathy, D., Zhou, C., Gope, D., Fedorov, I., Matas, R., Brooks, D., Wei, G.-Y., and Whatmough, P. Perfsage: Generalized inference performance predictor for arbitrary deep learning models on edge devices, 2023.

Dong, X. and Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

Duan, Y., Chen, X., Xu, H., Chen, Z., Liang, X., Zhang, T., and Li, Z. Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5251–5260, 2021.

Dudziak, L., Chau, T., Abdelfattah, M., Lee, R., Kim, H., and Lane, N. Brp-nas: Prediction-based nas using gcns. volume 33, pp. 10480–10490, 2020.

Elsken, T. Bosch gmbh. private communication., September 2023.

Kernighan, B. W. and Lin, S. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970. doi: 10.1002/j.1538-7305.1970.tb01770.x.

Krishnakumar, A., White, C., Zela, A., Tu, R., Safari, M., and Hutter, F. Nas-bench-suite-zero: Accelerating research on zero cost proxies. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Lahitani, A. R., Permanasari, A. E., and Setiawan, N. A. Cosine similarity to determine similarity measure: Study case in online essay assessment. In *2016 4th International Conference on Cyber and IT Service Management*, pp. 1–6, 2016. doi: 10.1109/CITSM.2016.7577578.

Lee, H., Hyung, E., and Hwang, S. J. Rapid neural architecture search by learning to generate graphs from datasets. In *International Conference on Learning Representations*, 2021a.

Lee, H., Lee, S., Chong, S., and Hwang, S. J. Help: Hardware-adaptive efficient latency prediction for nas via meta-learning. In *35th Conference on Neural Information Processing Systems (NeurIPS) 2021*. Conference on Neural Information Processing Systems (NeurIPS), 2021b.

Li, C., Yu, Z., Fu, Y., Zhang, Y., Zhao, Y., You, H., Yu, Q., Wang, Y., Hao, C., and Lin, Y. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=_0kaDkv3dVf.

Li, L. and Talwalkar, A. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

Lindauer, M. and Hutter, F. Best practices for scientific research on neural architecture search. *arXiv preprint arXiv:1909.02453*, 2019.

MacQueen, J. et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pp. 281–297. Oakland, CA, USA, 1967.

Ming Chen, Z. W., Zengfeng Huang, B. D., and Li, Y. Simple and deep graph convolutional networks. 2020.

Nair, S., Abbasi, S., Wong, A., and Shafiee, M. J. Mapleedge: A runtime latency predictor for edge devices. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3659–3667. IEEE, 2022.

Ning, X., Zhou, Z., Zhao, J., Zhao, T., Deng, Y., Tang, C., Liang, S., Yang, H., and Wang, Y. TA-GATES: An encoding scheme for neural network architectures. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=74fJwNrBlPI.

Ning, X., Zheng, Y., Zhou, Z., Zhao, T., Yang, H., and Wang, Y. A generic graph-based neural architecture encoding scheme with multifaceted information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(7): 7955–7969, 2023. doi: 10.1109/TPAMI.2022.3228604.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. pp. 2820–2828, 2019.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Wang, H., Wu, Z., Liu, Z., Cai, H., Zhu, L., Gan, C., and Han, S. Hat: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

White, C., Neiswanger, W., Nolen, S., and Savani, Y. A study on encodings for neural architecture search. In *Advances in Neural Information Processing Systems*, 2020.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

Xu, Y., Xie, L., Zhang, X., Chen, X., Shi, B., Tian, Q., and Xiong, H. Latency-aware differentiable neural architecture search. *arXiv preprint arXiv:2001.06392*, 2020.

Yan, S., Zheng, Y., Ao, W., Zeng, X., and Zhang, M. Does unsupervised architecture representation learning help neural architecture search? In *NeurIPS*, 2020.

Yan, S., Song, K., Liu, F., and Zhang, M. Cate: Computation-aware neural architecture encoding with transformers. In *ICML*, 2021.

Yang, A., Esperança, P. M., and Carlucci, F. M. Nas evaluation is frustratingly hard. 2020.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pp. 7105–7114. PMLR, 2019.

Yu, J., Jin, P., Liu, H., Bender, G., Kindermans, P.-J., Tan, M., Huang, T., Song, X., Pang, R., and Le, Q. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pp. 702–717. Springer, 2020.

Zela, A., Siems, J., Zimmer, L., Lukasik, J., Keuper, M., and Hutter, F. Surrogate nas benchmarks: Going beyond the limited search spaces of tabular nas benchmarks, 2020. URL https://arxiv.org/abs/2008.09777.