A Hardware-Aware Framework for Accelerating Neural Architecture Search Across Modalities

Daniel Cummings[1] **Anthony Sarah**[1] **Sharath Nittur Sridhar**[1] **Maciej Szankin**[1]
J. Pablo Muñoz[1] **Sairam Sundaresan**[1]

[1]Intel Labs, Intel Corporation

Abstract Recent advances in Neural Architecture Search (NAS) such as one-shot NAS offer the ability to extract specialized hardware-aware sub-network configurations from a task-specific super-network. While considerable effort has been employed towards improving the first stage, namely, the training of the super-network, the search for derivative high-performing sub-networks is still under-explored. We propose a flexible search framework that automatically and efficiently finds sub-networks that are optimized for different performance metrics and hardware configurations. Specifically, we demonstrate how various evolutionary algorithms when paired with lightly trained objective predictors can accelerate architecture search in a multi-objective setting for various modalities including machine translation, recommendation, and image classification.

1 Introduction

Artificial intelligence researchers are continually pushing the state-of-the-art in deep learning model performance across many application domains. Neural architecture search (NAS) has become an increasingly popular technique to achieve these performance gains with results that often outperform hand-designed architectures. In many cases, the deep neural network (DNN) design and evaluation process is tied to the hardware platform available to the researcher at the time (e.g., GPU). Furthermore, the researcher may have only been interested in a single performance objective such as accuracy when evaluating the network. Therefore, the network is inherently optimized for a specific hardware platform and specific objective. However, users wanting to solve the same problem for which the network was designed may have different hardware platforms available and may be interested in multiple performance metrics (e.g., accuracy *and* latency). To address this, we present a generalizable NAS framework that offers an assortment of search algorithms in a variety of objective optimization settings across several modalities. Additionally, we demonstrate how pairing evolutionary algorithms in an iterative fashion with lightly trained performance predictors can yield an accelerated and less costly exploration of a DNN architectural design space across the modalities of machine translation, recommendation, and image classification.

2 Methodology

Super-network or one-shot approaches to NAS have continued to grow in popularity for their training efficiency and the large range of inherent sub-networks configurations that make them attractive for hardware-aware latency optimization tasks. We focus this work on the sub-network search problem and describe a flexible search framework in Figure 1. For an arbitrary super-network reference architecture, modality, and task, our system flow automates the DNN architecture search process and discovers sub-networks that are optimal for a set of one or more performance objectives (e.g., accuracy, latency, MACs, etc.). Our framework includes both the pymoo (Blank and Deb, 2020) and Optuna (Akiba et al., 2019) optimization libraries for reproducibility and ease of future algorithmic enablement. The framework can efficiently interact with other NAS solutions for

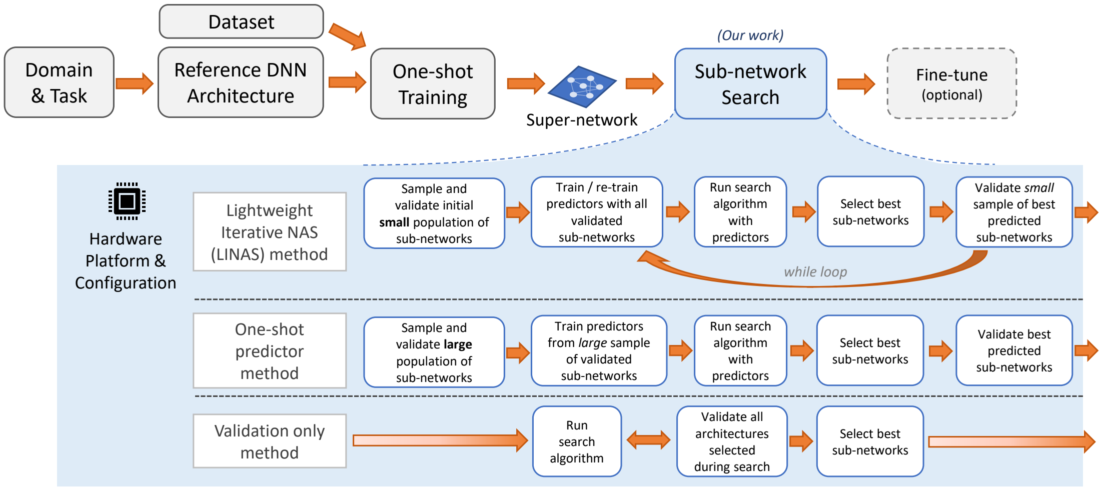

Dataset
Domain & Task
Reference DNN Architecture
One-shot Training
Super-network
(Our work)
Sub-network Search
Fine-tune (optional)
Hardware Platform & Configuration
Lightweight Iterative NAS (LINAS) method
Sample and validate initial small population of sub-networks
Train / re-train predictors with all validated sub-networks
Run search algorithm with predictors
Select best sub-networks
Validate small sample of best predicted sub-networks
while loop
One-shot predictor method
Sample and validate large population of sub-networks
Train predictors from large sample of validated sub-networks
Run search algorithm with predictors
Select best sub-networks
Validate best predicted sub-networks
Validation only method
Run search algorithm
Validate all architectures selected during search
Select best sub-networks

Figure 1: Generalizable framework for accelerating super-network type neural architecture search.

super-network generation and training (Munoz et al., 2022). The simplest *validation only* search method performs a validation measurement for sub-networks identified by the search algorithm of choice. However, even in the super-network NAS context, a validation measurement still requires a non-trivial amount of time and computational resources. A primary goal of this framework is to reduce the number of validation measurements (not predictions) that are required to find optimal DNN architectures given a set of performance objectives and a hardware platform. To this end, the framework also offers a *one-shot* predictor approach to reduce the validation cost overhead as described in existing work (Cai et al., 2019). Additionally, we describe a lightweight iterative NAS (LINAS) method in Appendix B that builds on the idea that lightly trained predictors can yield useful information for ranking sub-network configurations. The LINAS method increases the probability that optimal architectures will be identified in early stages of the search and avoids the upfront validation cost of the one-shot predictor approach. A short summary of LINAS (as in Figure 1) is as follows: using small validation populations (e.g., 10-50), we measure the objectives and store the results for each sub-network. These results are combined with any previous validation results and are used to train the objective predictors. We run a multi-objective algorithm search with a user defined evolutionary algorithm using that iteration's trained predictors and then use the Pareto optimal results to inform the next iterations population. Thus, as the search progresses, the predictors continue to improve, and the validated sub-networks quickly approach a more optimal multi-objective region.

The majority of NAS research efforts have focused on the computer vision task of image classification and only recently have other modalities, such as the rapidly growing field of language modeling or language translation, been investigated in detail (Feng et al., 2021; H. Wang et al., 2020; X. Zhang and Duh, 2020). Subsequently, understanding how NAS approaches generalize across modalities has not been studied in depth. In the study of our framework, our experiments encompass the modalities of image classification, machine translation, and recommendation as shown in Table 1. Additional details for each super-network are provided in Appendix D.

3 Experiments & Results

The main goal of our framework is to reduce the number of validation measurements required to find optimal DNN architectures in a multi-objective setting for any modality or domain-specific task. Specifically, we want to efficiently discover architectures with optimal trade-offs in high top-1 accuracy/BLEU/HR@10 and low latency. In the hardware-aware NAS context, latency is a

Table 1: Summary of super-networks and associated design search spaces used in the experiments.

Super-Network	Task	Dataset	Number Format	Objectives	Search Space Size (unique DNNs)
MobileNetV3	Image Classification	ImageNet	FP32	Top-1 Accuracy, Latency	$\sim 10^{19}$
ResNet50	Image Classification	ImageNet	FP32	Top-1 Accuracy, Latency	$\sim 10^{13}$
Transformer	Machine Translation	WMT 2014 En-De	FP32	BLEU Score, Latency	$\sim 10^{15}$
NCF	Recommendation	Pinterest-20	FP32	HR@10, Latency	$\sim 10^{7}$

highly important optimization objective since it directly relates to the real-time performance of a hardware system. Often MACs, FLOPs, or model parameter counts are used as an approximation of latency but do not guarantee correlation. We use the term "hardware-aware" to emphasize the focus on using latency as one of our main objectives but do not use any hardware architectural information to inform the search process. In this work we experiment on CPU, GPU, and mobile device platforms for evaluating our framework and we discuss the transferability of NAS results between CPU and GPU platforms in Appendix G.

Our framework makes extensive use of predictors to accelerate the discovery of optimal sub-networks, particularly when applying LINAS. Since validation evaluations of performance objectives, such as top-1 accuracy and latency, require a large amount of time, we follow the work in (Cai et al., 2019) and (H. Wang et al., 2020) and employ predictors. More specifically, we predict top-1 accuracy of sub-networks derived from MobileNetV3 and ResNet50 super-networks, hit ratio (HR@10) of sub-networks derived from NCF super-networks, bilingual evaluation understudy (BLEU) (Papineni et al., 2002) score of sub-networks derived from Transformer super-networks and latency of sub-networks derived from all super-networks.

Unlike prior work which use multi-layer perceptrons (MLPs) to perform prediction, we employ much simpler methods such as ridge regression, support vector machine regression (SVR) and stacked regression predictors. The authors of (Lu, Sreekumar, et al., 2021) and (Laube et al., 2022) have found that MLPs are inferior to other methods of prediction for low training example counts. We have found that these simpler methods converge more quickly and require both fewer training examples and much less hyper-parameter optimization than MLPs. The combination of performance objective prediction via simple predictors allows us to significantly accelerate the selection of sub-networks with minimal prediction error. As shown in Figure 2, the analysis of simple predictors is performed over a number of different trials to account for variance in the results. In each trial, the data set for each predictor is first split into train and test sets. Subsets of the train data set within the range of 100 to 1000 examples are used to train the predictor. For a given trial, the same test set with 500 examples is used to compute the prediction mean absolute percentage error (MAPE). This process is repeated for a total of 100 trials and the results averaged to compute the MAPE. See Section C for a detailed analysis of predictors for various objectives.

We start our experimental analysis using the MobileNetV3 super-network since it offers the largest search space size. It is crucial to note that a wide variety of evolutionary algorithms can be employed in our framework. These algorithms and other ablation studies are explored in Appendix G. However, for this section, the popular NSGA-II (Deb, Pratap, et al., 2002) algorithm is used as a baseline for both the validation-only and LINAS search approaches. Figures 3a and 3b illustrate the differences in how LINAS (with NSGA-II for the internal predictor loop), random search, and NSGA-II progress in the multi-objective search space. For the same evaluation count of 250, while NSGA-II begins progressing towards an optimal trade-off region, the LINAS results show how the

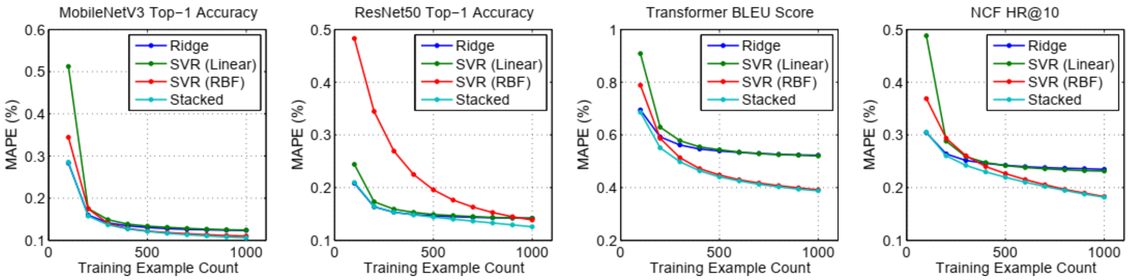
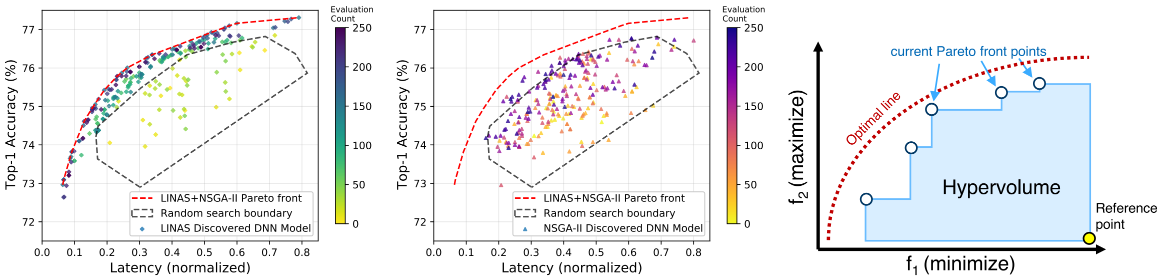


Figure 2: MAPE of predictors performing top-1 accuracy, BLEU score and HR@10 prediction versus the number of training examples for sub-networks derived from the super-networks shown in Table 1.



(a) LINAS Search.

(b) NSGA-II Search.

(c) Hypervolume example.

Figure 3: Search results in the MobileNetV3 search space (Titan-V GPU, batch size = 128) comparing (a): LINAS, and (b): NSGA-II approaches (algorithm settings in Table 5). Sub-figure (c) illustrates the hypervolume indicator.

exploration can be accelerated. Moreover, one can see how both approaches perform better than random search. In our experiments we use the term evaluation to refer to an actual validation measurement, not a predicted measurement. Since we are interested in evaluating the performance of various search algorithms in the multi-objective setting, we use the hypervolume indicator (Zitzler and Thiele, 1999) as shown in Figure 3c. When measuring two objectives, the hypervolume term represents the dominated *area* of the Pareto front.

A benefit of LINAS is that it can be used to run an accelerated NAS process without any prior knowledge of the latency range and we note that hardware platforms have very unique DNN latency characteristics. Figure 4 shows a consistent behavior for LINAS across GPU, CPU and mobile hardware settings. Depending on which region of the Pareto front is most important, an end-user would be more likely to identify optimal architectures in fewer evaluations with LINAS. Evaluation (validation measurement) counts directly correlate to the search time since the evolutionary algorithm runtime component is far smaller than evaluation runtimes (compute time breakdown given in Appendix F). When considering the performance of LINAS across various modalities as shown in Figure 5, a key observation is how differently LINAS accelerates to a better hypervolume versus the baseline NSGA-II and random search. Given the characteristics of the Transformer and NCF objective spaces, the LINAS result is less differentiated than in the image classification cases. We found that since the distribution of the sub-networks in these super-networks is both more constrained in range and occurs closer to an optimal region, one would be more likely to randomly find a good performing sub-network than in the MobileNetV3 or ResNet50 search spaces. Specifically, NCF is heavily biased towards matrix factorization as described by Rendle et al. (2020) and we discuss this effect in Appendix D.

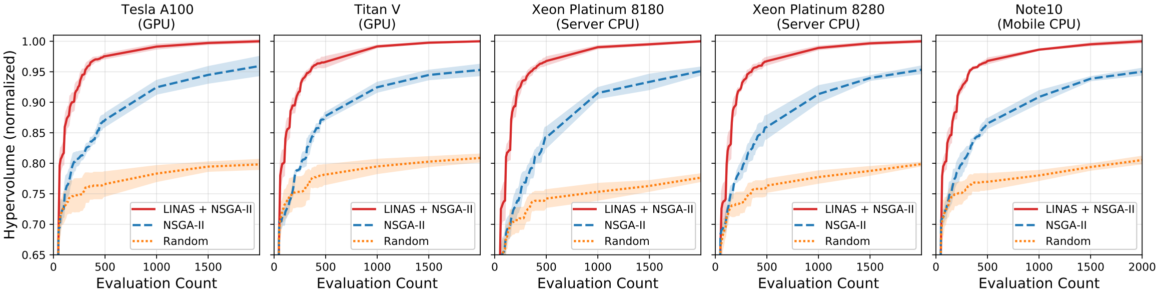
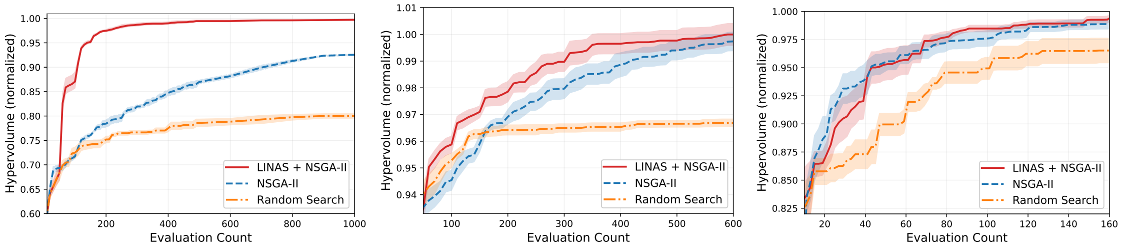


Figure 4: MobileNetV3 super-network search algorithm comparison on various hardware platforms. Shaded regions = standard error for 5 trials. Search parameter settings in Table 5.



(a) ResNet50

(b) Transformer

(c) NCF

Figure 5: Search algorithm comparison across modalities (Titan-V GPU).

4 Limitations and Broader Impact

From a framework perspective, our focus is limited to accelerating the post-training sub-network search process, not the optional fine-tuning stage. After promising DNN architectures are discovered with NAS, a user may achieve state-of-the-art performance for a particular performance bound (e.g., top-1 accuracy for a specific latency or MACs range) by finding the right combination of fine-tuning tactics or by completely re-training the sub-network from scratch (Wu et al., 2021). Moreover, we only focus on demonstrating evolutionary algorithm (EA) approaches that are known to pair well with weight-sharing-based search spaces. Genetic algorithms, a subset of EA, have been broadly applied for image classification NAS problems in both single-objective implementations (Guo et al., 2020) and in multi-objective approaches such as NSGA-Net (Lu, Whalen, et al., 2019). We note that reinforcement learning (RL) and gradient optimization have found success in the NAS field as well but do not evaluate the search performance of these methods (Liu, Simonyan, Vinyals, et al., 2017; Ren et al., 2021).

From a broader impact perspective, we do not anticipate that our work will have negative societal impacts. Our work leverages the one-shot weight sharing NAS paradigm which inherently provides massive DNN training savings in computation resources resulting in lower CO_2 emissions (Cai et al., 2019). Moreover, the computational cost (i.e., energy consumption) is further reduced by our approach to accelerate the architecture search process although it remains non-trivial.

5 Conclusion

We have proposed and demonstrated a flexible NAS framework that efficiently finds diverse sets of sub-networks in the hardware-aware multi-objective setting using super-networks from various modalities. As NAS research continues to gain momentum, we highlight the need to continue to investigate the generalizability of NAS approaches in modalities and tasks outside of computer vision. Future work will include extending the experiments to a larger variety of hardware platforms and DNN application domains.

References

Akiba, T. et al. (2019). “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Bender, G. et al. (July 2018). “Understanding and Simplifying One-Shot Architecture Search”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 550–559. URL: https://proceedings.mlr.press/v80/bender18a.html.

Blank, J. and K. Deb (2020). “pymoo: Multi-Objective Optimization in Python”. In: *IEEE Access* 8, pp. 89497–89509.

Blank, J., K. Deb, et al. (2021). “Generating Well-Spaced Points on a Unit Simplex for Evolutionary Many-Objective Optimization”. In: *IEEE Transactions on Evolutionary Computation* 25.1, pp. 48–60.

Cai, H. et al. (2019). “Once-for-all: Train one network and specialize it for efficient deployment”. In: *arXiv preprint arXiv:1908.09791*.

Deb, K., A. Pratap, et al. (2002). “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2, pp. 182–197.

Deb, K. and J. Sundar (2006). “Reference Point Based Multi-Objective Optimization Using Evolutionary Algorithms”. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’06. Seattle, Washington, USA: Association for Computing Machinery, pp. 635–642. URL: https://doi.org/10.1145/1143997.1144112.

Deng, J. et al. (2009). “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255.

Elsken, T., J. H. Metzen, and F. Hutter (2019). *Neural Architecture Search: A Survey*. arXiv: 1808.05377 [stat.ML].

Feng, B., D. Liu, and Y. Sun (2021). “Evolving Transformer Architecture for Neural Machine Translation”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: Association for Computing Machinery, pp. 273–274. URL: https://doi.org/10.1145/3449726.3459441.

Guo, Z. et al. (2020). *Single Path One-Shot Neural Architecture Search with Uniform Sampling*. arXiv: 1904.00420 [cs.CV].

He, K. et al. (June 2016). “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: http://dx.doi.org/10.1109/cvpr.2016.90.

He, X. et al. (2017). “Neural collaborative filtering”. In: *Proceedings of the 26th international conference on world wide web*, pp. 173–182.

Howard, A. et al. (2019). “Searching for mobilenetv3”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324.

Laube, K. A., M. Mutschler, and A. Zell (2022). *What to expect of hardware metric predictors in NAS*. URL: https://openreview.net/forum?id=2DJn3E7lXu.

Li, K. et al. (2019). “Two-Archive Evolutionary Algorithm for Constrained Multiobjective Optimization”. In: *IEEE Transactions on Evolutionary Computation* 23.2, pp. 303–315.

Liu, H., K. Simonyan, O. Vinyals, et al. (2017). “Hierarchical Representations for Efficient Architecture Search”. In: *CoRR* abs/1711.00436. arXiv: 1711.00436. URL: http://arxiv.org/abs/1711.00436.

Liu, H., K. Simonyan, and Y. Yang (2018). “DARTS: Differentiable Architecture Search”. In: *CoRR* abs/1806.09055. arXiv: 1806.09055. URL: http://arxiv.org/abs/1806.09055.

Lu, Z., K. Deb, et al. (2020). *NSGANetV2: Evolutionary Multi-Objective Surrogate-Assisted Neural Architecture Search*. arXiv: 2007.10396 [cs.CV].

Lu, Z., G. Sreekumar, et al. (Sept. 2021). “Neural Architecture Transfer”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.9, pp. 2971–2989. URL: `http://dx.doi.org/10.1109/TPAMI.2021.3052758`.

Lu, Z., I. Whalen, et al. (2019). *NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm.* arXiv: `1810.03522 [cs.CV]`.

Munoz, J. P. et al. (2022). “Automated Super-Network Generation for Scalable Neural Architecture Search”. In: *First Conference on Automated Machine Learning (Main Track)*. URL: `https://openreview.net/forum?id=HK-zmbTB8gq`.

Muñoz, J. P. et al. (2021). “Enabling NAS with Automated Super-Network Generation”. In: *CoRR* abs/2112.10878. arXiv: `2112.10878`. URL: `https://arxiv.org/abs/2112.10878`.

Ozaki, Y. et al. (2020). “Multiobjective Tree-Structured Parzen Estimator for Computationally Expensive Optimization Problems”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO ’20. Cancún, Mexico: Association for Computing Machinery, pp. 533–541. URL: `https://doi.org/10.1145/3377930.3389817`.

Panichella, A. (2019). “An Adaptive Evolutionary Algorithm Based on Non-Euclidean Geometry for Many-Objective Optimization”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’19. Prague, Czech Republic: Association for Computing Machinery, pp. 595–603. URL: `https://doi.org/10.1145/3321707.3321839`.

Papineni, K. et al. (Oct. 2002). “BLEU: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318.

Ren, P. et al. (2021). *A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions.* arXiv: `2006.02903 [cs.LG]`.

Rendle, S. et al. (2020). “Neural collaborative filtering vs. matrix factorization revisited”. In: *Fourteenth ACM conference on recommender systems*, pp. 240–248.

Vaswani, A. et al. (2017). *Attention Is All You Need.* arXiv: `1706.03762 [cs.CL]`.

Wang, H. et al. (2020). “Hat: Hardware-aware transformers for efficient natural language processing”. In: *arXiv preprint arXiv:2005.14187*.

Wu, J. et al. (2021). *Stronger NAS with Weaker Predictors.* arXiv: `2102.10490 [cs.LG]`.

Zhang, Q. and H. Li (2007). “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition”. In: *IEEE Transactions on Evolutionary Computation* 11.6, pp. 712–731.

Zhang, X. and K. Duh (2020). “Reproducible and Efficient Benchmarks for Hyperparameter Optimization of Neural Machine Translation Systems”. In: *Transactions of the Association for Computational Linguistics* 8, pp. 393–408. URL: `https://aclanthology.org/2020.tacl-1.26`.

Zitzler, E. and L. Thiele (1999). “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach”. In: *IEEE transactions on Evolutionary Computation* 3.4, pp. 257–271.

A Additional Background and Related Work

The computational overhead of evaluating DNN architectures during NAS can be very costly due to the training and validation cycles. To address the training overhead, novel weight-sharing approaches known as one-shot or super-networks (Bender et al., 2018; Cai et al., 2019; Liu, Simonyan, and Yang, 2018) have offered a way to mitigate the training overhead by reducing training times from thousands to a few GPU days (Elsken et al., 2019). These approaches train a task-specific super-network architecture with a weight-sharing mechanism that allows the sub-networks to be treated as unique individual architectures. This enables sub-network model extraction and validation without a separate training cycle. However, the validation component still comes with a high overhead since there are many possible sub-networks which may be found from large super-networks (e.g., search space size of $\sim 10^{19}$) and the validation step itself comes with a computational cost, especially for larger datasets such as ImageNet (Deng et al., 2009). One popular way to mitigate the validation cost in one-shot networks is to train predictors for objectives such as inference time (a.k.a. latency) and accuracy from a training set with thousands of sampled architectures (Cai et al., 2019).

Similar to the LINAS formulation, there are approaches that iterate training and search in order to fine-tune the super-network during training (Lu, Sreekumar, et al., 2021). However, the fine-tuning of these approaches is influenced by the hardware platform used during search which could require training to be redone if the resulting super-network were deployed on a different platform (e.g., trained on GPU, deployed onto Raspberry Pi which has very different latency characteristics.). Similarly, work by Wu et al. (2021) uses a simple Bayesian optimization paired with predictors to iteratively direct the search. We highlight the feedback from an external reviewer following the submission of this work that the LINAS formulation has strong similarities to NSGANetV2-MSuNAS from Lu, Deb, et al. (2020) to which the authors were unaware of at the time. This work extends past the scope of NSGANetV2 in that we evaluate many evolutionary algorithms, add additional modalities, and run new ablations that give additional insight into this paradigm of NAS.

B LINAS Algorithm

One of the primary goals for our framework is to reduce the number of validation measurements that are required to find optimal DNN architectures in a multi-objective search space that works well across modalities. While related work shows that using trained predictors can speed up the DNN architecture search process, there remains a substantial cost to training the predictors since the number of validated training samples can range between 1000 and 16,000. Interestingly, as discussed in detail in Appendix C, simple accuracy predictors can achieve acceptable mean absolute percentage error (MAPE) with far fewer training samples. We build on this insight that lightly trained predictors can offer a useful surrogate signal during search. Algorithm 1 describes our generalizable Lightweight Iterative NAS (LINAS) method. We first randomly sample the architecture search space to serve as the initial validation population. For each sub-network in the validation population, we measure each objective and store the result. These results are combined with all previous validation results and are used to train the objective predictors. For each iteration, we run a multi-objective algorithm search using that iteration's trained predictors for a high number of generations (e.g., > 100) to allow the algorithm to explore the predicted objective space sufficiently. This predictor-based search runs very quickly since no validation measurements occur. Finally, we select the most optimal population of diverse DNN architectures from the predictor-based search to add to the next validation population, which then informs the next round of predictor training. This cycle continues until the iteration count limit is met or an end-user decides a sufficient set of architectures has been discovered. We note that the LINAS approach can be applied with any single-, multi-, or many-objective evolutionary algorithms (EA) and generalizes to work with any super-network framework. Additionally, it allows for the interchanging of algorithms, tuning parameters (e.g., crossover, mutation, population), and predictor types for each iteration.

Algorithm 1 Generalizable Lightweight Iterative Neural Architecture Search (LINAS)

```
**Input**: Objectives $f_m$, super-network with weights $\mathcal{W}$ and configurations $\Omega$, predictor model for each objective $Y_m$, LINAS population $P$ size $n$, number of LINAS iterations $I$, evolutionary algorithm $\mathcal{E}$ with number of evaluations $J$.
$P_{i=0} \longleftarrow \{\omega_n\} \in \Omega$ // sample $n$ sub-networks for first population
**while** $i++ < I$ **do**
    $D_{i,m} \longleftarrow f_m(P_i \in \Omega; \mathcal{W})$ // measure objectives $f_m$, store data $D_{i,m}$
    $D_{all,m} \longleftarrow D_{all,m} \cup D_{i,m}$
    $Y_{m,pred} \longleftarrow Y_{m,train}(D_{all,m})$ // train predictors for each objective
    **while** $j++ < J$ **do**
        $P_{\mathcal{E}_j} \longleftarrow \mathcal{E}(Y_{m,pred}, j)$ // run $\mathcal{E}$ for $J$ evaluations
    **end while**
    $P_i \longleftarrow P_{\mathcal{E},best\_unique} \in P_{\mathcal{E}_J}$ // retrieve optimal and unique population of sub-networks
**end while**
**Output**: All validated sub-networks configurations $P_I$, predictor search results $P_{\mathcal{E}_{I,J}}$, and validation data $D_{all,m}$.
```

C Predictors

As described in Figure 1, our work makes extensive use of predictors to accelerate the selection of sub-networks, particularly when applying LINAS. Since validation evaluations of performance objectives, such as top-1 accuracy and latency, require a large amount of time, we follow the work in (Cai et al., 2019) and (H. Wang et al., 2020) and employ predictors. More specifically, we predict top-1 accuracy of sub-networks derived from MobileNetV3 and ResNet50 super-networks, hit ratio (HR@10) of sub-networks derived from NCF super-networks, bilingual evaluation understudy (BLEU) (Papineni et al., 2002) score of sub-networks derived from Transformer super-networks and latency of sub-networks derived from all super-networks. Predictors are necessary since performing actual measurements of performance objectives such as accuracy or latency would be prohibitively slow. In light of their importance, a better understanding of their performance is needed.

The analysis of predictors is performed over a number of different trials to account for variance in the results. In each trial, the data set for each predictor is first split into train and test sets. Subsets of the train data set within the range of 100 to 1000 examples are used to train the predictor. For a given trial, the *same* test set with 500 examples is used to compute the prediction mean absolute percentage error (MAPE). This process is repeated for a total of 100 trials and the results averaged to compute the MAPE shown in Figure 6.

The top row of Figure 6 shows the MAPE of different predictors for each super-network type in Table 1. The *stacked* predictor is a combination of ridge and SVR (RBF) regressors which "stacks" the predictions from each of these two regressors and uses them as the input to a final ridge regressor. The bottom row shows the correlation between actual and predicted values after training the stacked predictor with 1000 examples. Note that the Kendall rank correlation coefficient τ is also shown for each case. In all cases, these simple predictors provide small error (maximum MAPE of 0.91%) and high correlation (minimum τ of 0.8348) with actual values.

The analysis of latency prediction is performed similarly to the top-1 accuracy, BLEU score and HR@10 approach with the results shown in Figure 7. The top row shows the MAPE of different predictors for each super-network type. The bottom row shows the correlation between actual and

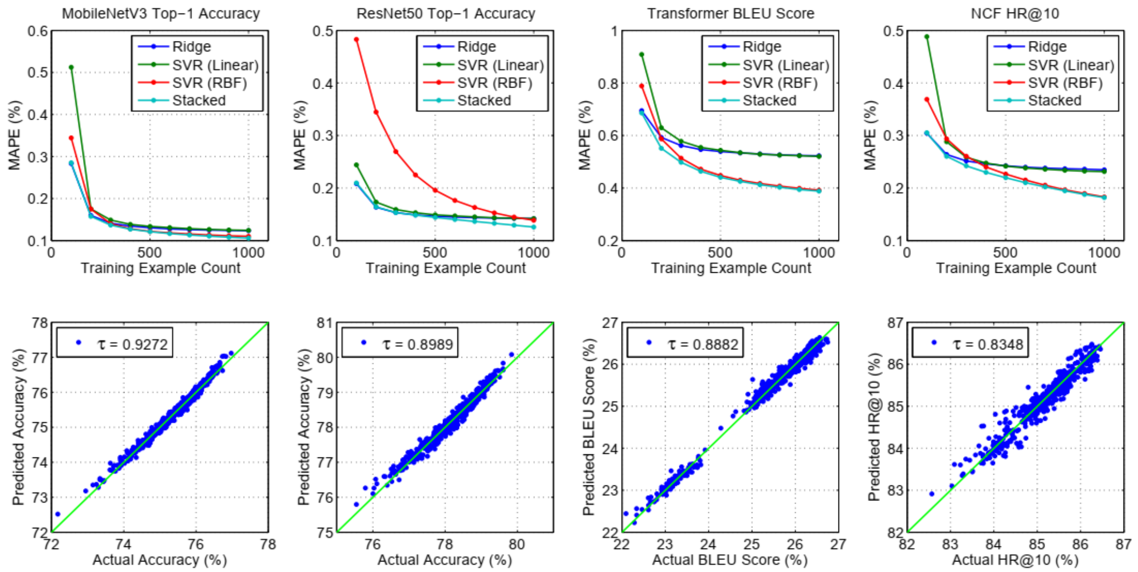


Figure 6: MAPE of predictors performing top-1 accuracy, BLEU score and HR@10 prediction versus the number of training examples for sub-networks derived from the super-networks shown in Table 1 (top row). Correlation and Kendall τ coefficient between actual and predicted values after training the stacked predictor with 1000 examples (bottom row). The ideal correlation is shown by the green line.

predicted latencies after training the stacked predictor with 1000 examples. Note that the Kendall rank correlation coefficient τ is also shown for each case.

D Super-network Details

For the modality of image classification, we leverage two super-networks derived from MobileNetV3 (Howard et al., 2019) and ResNet50 (K. He et al., 2016) which are described in Once-for-all (OFA) (Cai et al., 2019). OFA employs a progressive shrinking method during super-network training resulting in *elastic* design parameters that can represent the full architectural search space. For additional variety in this domain, we use recent work by Munoz et al. (2022) and Muñoz et al. (2021), called BootstrapNAS, and discuss the sub-network search process for the quantized INT8 space in the Appendix G.5.

Super-network approaches have recently been applied in the domain of Natural Language Processing (NLP). Hardware-aware Transformers (HAT) (H. Wang et al., 2020) achieve this goal by extending the network elasticity type of weight sharing approach to this domain. In HAT, the authors introduce *arbitrary encoder-decoder attention*, to break the information bottleneck between the encoder and decoder layers in Transformers (Vaswani et al., 2017). Additionally, they propose heterogenous transformer layers to allow for different layers to have different parameters.

Neural Collaborative Filtering (NCF) (X. He et al., 2017), a popular method for recommendation problems, combines the benefits of traditional matrix factorization and fully connected neural networks. We adapt this model architecture into an elastic training framework similar to HAT wherein each embedding layer and dense layer is fully elastic.

D.1 Search Space Encoding

A key consideration of the super-network NAS process is encoding a representation of the architectural design variables in a way that is useful for the search algorithms. For illustration, we

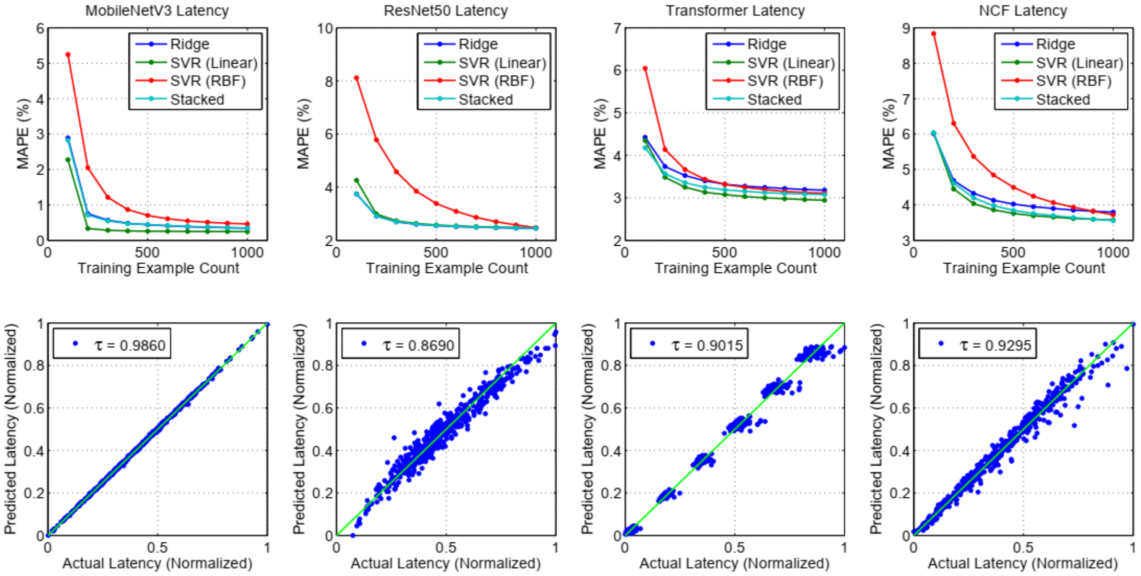
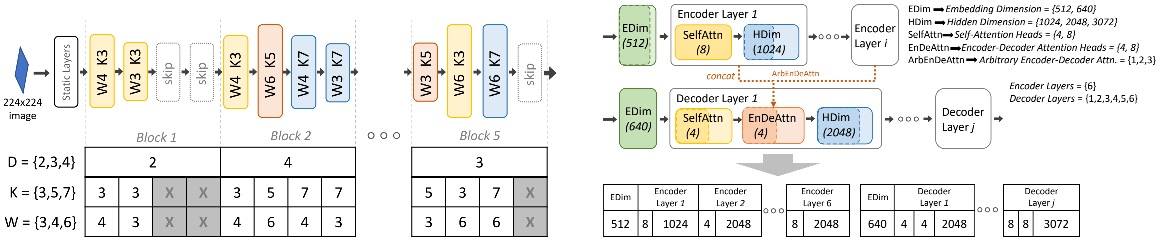


Figure 7: MAPE of predictors performing latency prediction versus the number of training examples for sub-networks derived from the super-networks shown in Table 1 (top row). Correlation and Kendall τ coefficient between actual and predicted latencies after training the stacked predictor with 1000 examples (bottom row). The ideal correlation is shown by the green line.



(a) MobileNetV3 design variable encoding.

(b) Transformer design variable encoding.

Figure 8: Super-network encoding strategies for MobileNetV3 and Transformer each having 45 and 40 design variables respectively.

summarize our encoding strategy for the MobileNetV3 and Transformer super-networks in Figure 8. By mapping each super-network architecture design variable to several integer options, this search space encoding offers a compatible interface for the evolutionary operators (e.g., mutation, crossover, etc.).

D.2 MobileNetV3 Details

For the image classification task with MobileNetV3, we experiment on the ImageNet validation dataset (Deng et al., 2009) and use the pre-trained super-network weights from *ofa_mbv3_d234_e346_k357_w1.0*[1], trained with progressive shrinking. For the architecture design variables, we allow for an elastic layer depth chosen from [2, 3, 4], an elastic width expansion ratio chosen from [3, 4, 6], an elastic kernel size chosen from [3, 5, 7], and use an input image resolution of 224x224. The layer depth can affect the mapping of the kernel size and expansion ratio design variables as shown in Figure 8(a). For more details on this super-network please refer to the work by Cai et al. (2019).

[1] https://github.com/mit-han-lab/once-for-all

D.3 ResNet50 Details

For the image classification task with ResNet50, we use the ImageNet validation dataset (Deng et al., 2009) and use the pre-trained super-network weights from *ofa_resnet50_d=0+1+2_e=0.2+0.25+0.35_w=0.65+0.8+1.0*[1], trained with progressive shrinking for our experiments. For the architecture design variables, we allow for an elastic layer depth chosen from [0, 1, 2], an elastic width expansion ratio chosen from [0.65, 0.8, 1.0], an elastic expansion ratio chosen from [0.2, 0.25, 0.35], and use an input image resolution of 224x224.

D.4 Transformer Details

For the machine translation task, we mainly experiment on the WMT 2014 En-De data set. We follow a similar pre-processing technique proposed in (H. Wang et al., 2020) for the data. Similar to (H. Wang et al., 2020), we use the search space with an embedding dimension chosen from [512, 640] , hidden dimension from [1024, 2048, 3072] , attention head number from [4, 8], decoder layer number from [1, 2, 3, 4, 5, 6] and a constant encoder layer number of [6]. In (H. Wang et al., 2020), although the authors use the inherited weights from the Transformer super-network, for the evolutionary search, they re-train the sub-networks from scratch in the final results. In our results, we do not re-train these networks from scratch. Additionally, we train the predictor directly on the *bilingual evaluation understudy (BLEU)* (Papineni et al., 2002) score. For the BLEU score evaluation, we use a beam size of 5 and a length penalty of 0.6.

D.5 NCF Details

For the recommendation task, we experiment on the Pinterest-20 dataset and follow a similar pre-processing technique used in (X. He et al., 2017) . We use the *Neural Matrix Factorization* (NeuMF) model from (X. He et al., 2017), which is a fusion of *Generalized Matrix Factorization* (GMF) and *Multi-Layer Perceptron* (MLP). We create an elastic NCF super-network model with the embedding dimension for MLP and GMF layers sampled from [8, 16,32,64, 128], MLP layer number from [1,2,3,4,5,6], and MLP hidden sizes from [8, 16, 32, 64, 128, 256, 512, 1024]. We train the NCF super-network by uniformly sampling different sub-networks for each mini-batch of training.

In our experiments we see that when the matrix factorization module of the NCF sub-network was sufficiently large, the results of the subnetwork were dominated by it versus the MLP module. Rendle et al. (2020) substantiate this hypothesis in their paper by showing that a well-tuned matrix factorization approach can substantially outperform proposed learned similarities such as an MLP. We thus attribute the diminished improvement in the performance of LINAS on NCF to the degeneracies in the search space caused by a more powerful matrix factorization module which strongly dominates the HR@10.

E Latency Measurements

Since our experiments measure latency values from different manufacturers and there are possible proprietary issues in sharing what could be perceived as official benchmark data, we normalize latency results to be within $[0, 1]$. More specifically, the normalized latency $\hat{l}$ is given by $\hat{l} = \frac{l-l_{min}}{l_{max}} \in [0, 1]$ where l is the unnormalized latency, l_{min} is the minimum unnormalized latency and l_{max} is the maximum unnormalized latency. Using normalized latency does not change the underlying search results that are demonstrated. For comparative latency performance metrics related to our test platforms, we point the reader to the MLCommons[2] benchmark suite.

[2] https://mlcommons.org

F Test Platforms and Compute Time

We use both CPU and GPU platforms for running our experiments. The hardware platforms and their characteristics are shown in Table 2. For the Note10 mobile CPU experiment shown in Figure 4, we use a latency look-up table provided by Cai et al., 2019 since we did not have direct access to that platform.

Table 2: Hardware platforms used for NAS experimentation.

Name	Memory	Thread Count (Host CPU)	Microarchitecture (Host CPU)
Intel® Xeon® Platinum 8180	192 GB	56	Skylake (SKX)
Intel® Xeon® Platinum 8280	192 GB	56	Cascade Lake (CLX)
NVIDIA® Titan V®	32 GB	32	Cascade Lake (CLX)
NVIDIA® Tesla® V100	32 GB	32	Skylake (SKX)
NVIDIA® Tesla® A100	32 GB	32	Cascade Lake (CLX)

In terms of GPU wall clock time required to perform search, we note that there is a wide range of results that would be dependent on the supporting hardware platform configuration. For example, for MobileNetV3, a sub-network search with 2000 evaluations would take approximately 9.5 GPU hours with an evolutionary algorithm run time on the order of minutes. Because the evolutionary algorithm run times are extremely small when compared to validation measurement run times, we view the evaluation count (e.g., Figure 5) as a more universal metric of search time efficiency in this work.

To provide more insights into time complexity of presented algorithms, an extensive set of tests was performed to measure wall-clock time of each algorithm needed to achieve a certain hypervolume threshold on each of the search spaces presented in this work. For each super-network, the hypervolume thresholds were selected based on the maximum hypervolume achieved by random search and NSGA-II for a given search space, respectively. In the latter case, the results for random search are not shown as it never achieved the given hypervolume level within a set number of evaluations. Table 3 shows detailed information on how much time was spent on the model evaluation and the search process itself.

G Additional Results

G.1 One-Shot Predictor Comparison

Since MobileNetV3 is a Once-for-All (OFA) super-network, we run the genetic algorithm (GA) search as used in the OFA paper and show the results in Figure 9 for comparison. This approach follows the one-shot predictor method as shown in Figure 1 where predictors for the objectives are trained with 1000 samples before the search starts in this setup. The search then runs a large amount of predictor-based evaluations in the latency range of interest. The intent of the OFA GA search algorithm is to maximize the accuracy for a particular latency constraint. In the multi-objective setting this has a few limitations such as needing prior knowledge of the latency space and requiring a user to manually define separate search groups across the known latency range that are unique to each hardware platform. In our Figure 9 example we define four search groups (each with unique latency constraints) and note that LINAS only requires 250 evaluations to find a more diverse Pareto front compared to the GA search from the OFA paper which uses 1000 evaluations to build predictors. A key takeaway is that LINAS can be used to extend the search capabilities of any super-network or weight-sharing NAS framework in the multi-objective setting.

Table 3: Comparison of algorithms and their average run time on all presented search spaces to a given normalized hypervolume (HV) threshold based on a platform with NVIDIA® Titan V® (evaluation) and Intel® Xeon® Platinum 8280 (search).

Super-Network	Search Algorithm	Evaluations	Evaluation Cost (GPU Hours)	Search Cost (CPU Hours)	Total Cost (Hours)
MobileNetV3	Normalized HV = 0.810				
	LINAS + NSGA-II	100	0.479	0.0095	0.489
	NSGA-II	260	1.247	0.0014	1.248
	Random	2000	9.593	0.0017	9.594
	Normalized HV = 0.955				
	LINAS + NSGA-II	346	1.746	0.0331	1.779
	NSGA-II	2000	9.593	0.0017	9.594
	Random	—	—	—	—
ResNet50	Normalized HV = 0.800				
	LINAS + NSGA-II	57	0.545	0.0047	0.549
	NSGA-II	255	2.438	0.0014	2.439
	Random	1000	9.559	0.0016	9.560
	Normalized HV = 0.925				
	LINAS + NSGA-II	155	1.481	0.0142	1.496
	NSGA-II	1000	9.559	0.0016	9.560
	Random	—	—	—	—
Transformer	Normalized HV = 0.967				
	LINAS + NSGA-II	111	1.886	0.0035	1.890
	NSGA-II	191	3.246	0.0014	3.248
	Random	600	10.197	0.0015	10.199
	Normalized HV = 0.997				
	LINAS + NSGA-II	465	7.903	0.0156	7.918
	NSGA-II	600	10.197	0.0015	10.199
	Random	—	—	—	—
NCF	Normalized HV = 0.965				
	LINAS + NSGA-II	87	2.884	0.0138	2.898
	NSGA-II	89	2.950	0.0014	2.952
	Random	160	5.304	0.0014	5.305
	Normalized HV = 0.989				
	LINAS + NSGA-II	148	4.906	0.0241	4.930
	NSGA-II	160	5.304	0.0014	5.305
	Random	—	—	—	—

G.2 LINAS Ablation Studies

LINAS offers a great deal of flexibility in terms predictor and algorithm options for the internal loop. Figure 10a shows a LINAS specific ablation study using the various EA algorithms (discussed in Appendix H.2) for the internal predictor loop including the performance of various algorithms without LINAS. We find that Pareto based MOEAs such as NSGA-II (Deb, Pratap, et al., 2002) and AGE-MOEA (Panichella, 2019) and the indicator based U-NSGA-III (Deb and Sundar, 2006) perform well for this task. MOTPE Ozaki et al. (2020) by itself finds good sub-networks in the very early stage of the search process but suffers from very high run-times for evaluation counts above 500. This limits the ability of MOTPE to efficiently be used in the LINAS internal predictor loop since it will not approach the near-optimal Pareto region in the predictor space.

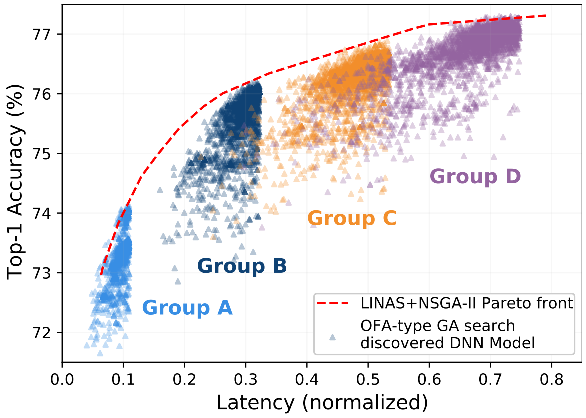
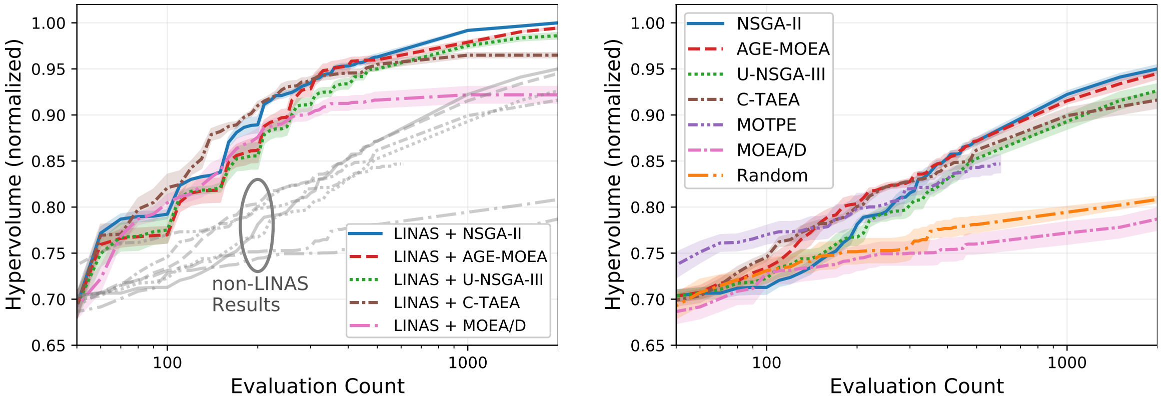


Figure 9: Results for the GA approach used in the OFA paper that uses predictors trained from 1000 evaluations where we run four different latency constrained searches. Triangle symbols represent predicted performance whereas the dotted red line represent the actual Pareto front from a LINAS+NSGA-II with 250 evaluations.

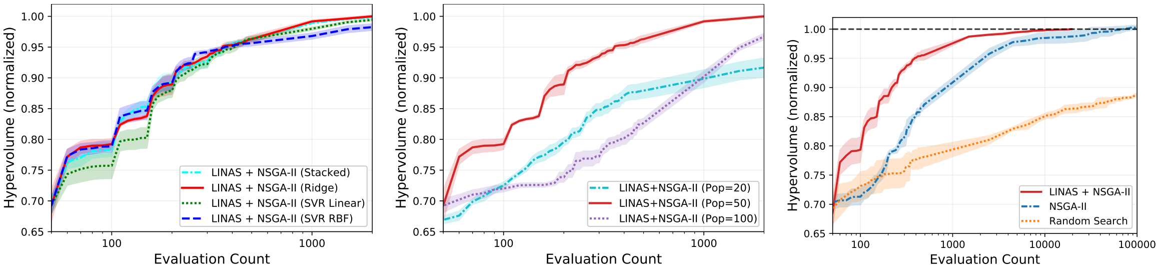


(a) LINAS comparison for various EAs in the predictor loop.

(b) Search algorithm comparison without using LINAS.

Figure 10: Comparison of search algorithms in the MobileNetV3 design space for hypervolume (top-1 accuracy and latency) versus evaluation count. Shaded regions show the standard error for 5 trials with different random seeds. Search parameter settings in Appendix H.2.

Figure 11a highlights that the choice of the underlying predictor algorithm has little impact on the performance of LINAS. Next, in Figure 11b we compare various LINAS runs with different population sizes where a population of 50 gives the best performance for the MobileNetV3 super-network. Finally, we note that while the intent of LINAS is to run for the fewest number of evaluations as possible, an extended run shows that it would take NSGA-II a significant amount of evaluations to catch up with the LINAS hypervolume at 20,000 evaluations. For the subsequent experiments and consistency, we compare LINAS (with NSGA-II for the internal predictor loop) against validation-only measurements from a random search that uniformly samples the architecture space and NSGA-II itself using the algorithm and the predictor settings in Table 5.



(a) Predictor ablation study. (b) Population size study. (c) Extended run example

Figure 11: Hypervolume ablations studies on MobileNetV3 (Titan-V GPU, batch=128). Shaded regions show the standard error for 5 trials with different random seeds.

G.3 Hardware Platform Transferability

One of the key goals of our framework is to accelerate the sub-network search process to address the issue that every hardware platform and/or configuration has unique latency characteristics and therefore unique optimal sub-networks in their respective multi-objective search spaces. To illustrate this behavior, we use the MobileNetV3 super-network where Figure 12 shows that an optimal set of sub-networks found on a CPU platform may not transfer to the optimal objective region on a GPU platform and vice versa. Furthermore, within a hardware platform, Figure 13 shows that sub-network configurations found to be optimal to one CPU hardware configuration (e.g., batch size = 1, thread count = 1), do not transfer optimally to other hardware batch size and thread count configurations.

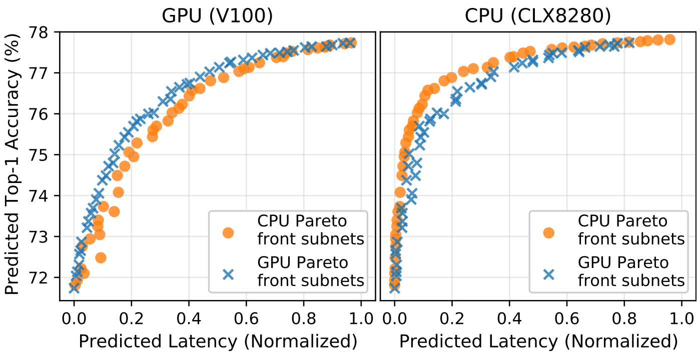


Figure 12: MobileNetV3 Pareto fronts specialized to GPU (V100) and CPU (CLX) showing that optimal sub-network configurations found on one hardware platform do not translate to the optimal sub-networks for another. Batch size was 128.

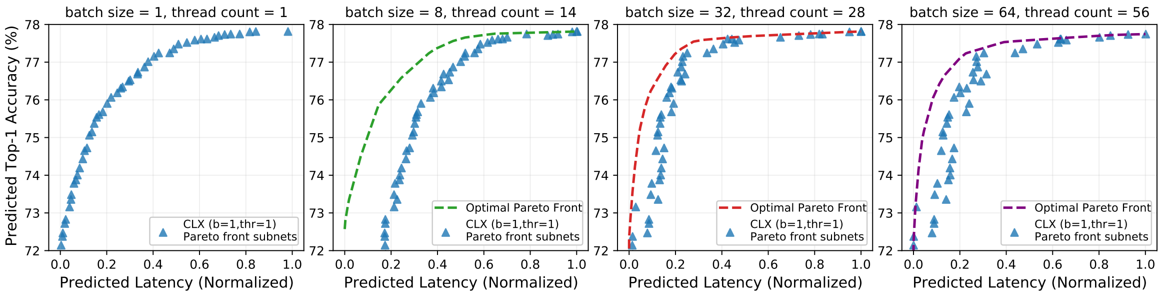
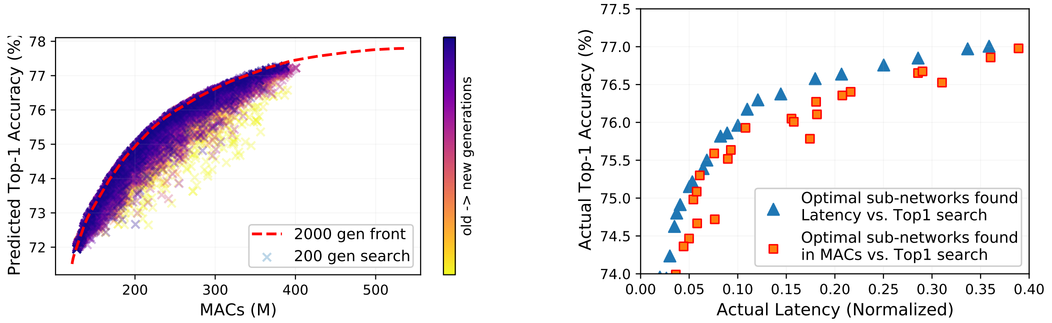


Figure 13: MobileNetV3 Pareto fronts with CLX for specialized thread counts/batch sizes, and the non-specialized configurations for comparison.

G.4 Multiply-Accumulates to Latency

In addition to evaluating search performance on the latency, accuracy (Top-1), and BLEU score objectives, we looked at the search trends in terms of multiply-accumulates (MACs) and accuracy as shown in Figure 14a using the fvcore[3] library. Often, multiply-accumulates (MACs) or floating point operations per second (FLOPs) are used to approximate latency. However, we note that the transferrability between these metrics has its limitations. For example, Figure 14b highlights that optimal sub-networks identified during a lengthy (e.g., run search until the Pareto front is saturated with sub-network options) multi-objective MACs and top-1 accuracy NSGA-II search do not translate to the most optimal sub-networks identified during a latency-based NSGA-II search. One benefit of a MACs search is that the best Pareto front population would be ideal for a warm-start population on subsequent searches for a given super-network. Another option in our framework would be to perform a many-objective search (e.g., U-NSGA-III) to find optimal sub-networks in the latency, accuracy, *and* MACs search space.



(a) NSGA-II search progression in the MACs versus top-1 accuracy objective space.

(b) MACs-based optimal sub-networks placed in the latency space. CLX CPU with batch=128, threads=56.

Figure 14: Comparison between latency and MACs-based NSGA-II searches using the MobileNetV3 super-network showing that the best Pareto front sub-networks from a MACs-based search do not always translate optimally to the latency objective space.

G.5 LINAS Performance for Quantized Super-Networks

In addition to searching for optimal configurations using the OFA image classification super-networks (based in FP32 number format), we also experimented with finding optimal INT8 models from the novel BootstrapNAS (BNAS) ResNet50 super-network (Munoz et al., 2022; Muñoz et al., 2021). BNAS transforms a single reference pre-trained DNN architecture into a super-network and streamlines the sub-network search process in the quantized INT8 space. Specifically, we leverage a BNAS ResNet50 super-network (BNAS-ResNet50Q) that has a different design space, specified with different values for the search design variables, than the Once-for-all ResNet50 model discussed in Section D.3. In this setup, the elastic layer depth chosen from [0, 1], an elastic width expansion ratio chosen from [0.65, 0.8, 1.0] and an elastic expansion ratio chosen from [0.2, 0.25].

Our experiment follows the steps outlined in Figure 1 with an additional weight conversion from FP32 to INT8 and standardized fine-tuning of the INT8 model in the last step. As shown on the Figure 15, LINAS offers improvements in terms of hypervolume progression and in the time required to find diverse models in the Pareto front. NSGA-II under-performs in early stages of the search when compared to random search, which could be explained by the characteristics of the BootstrapNAS super-network, and its smaller selection of elastic parameters that have been limited to those promising better performance for the extracted sub-networks. As shown on the Figure 16,

[3] https://github.com/facebookresearch/fvcore

the overall distribution of the randomly sampled configurations is closer to the optimal region of the objective space, which may be the cause of random search yielding comparable results in the early stages of the search process. This result is likely due to the nature of BNAS' process for selecting promising elastic design parameters and also that the search space derived by BNAS-ResNet50Q ($\sim 10^7$) is much smaller than OFA's ResNet-50 ($\sim 10^{13}$).

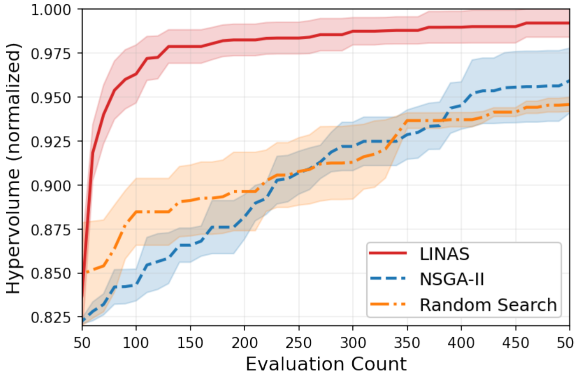
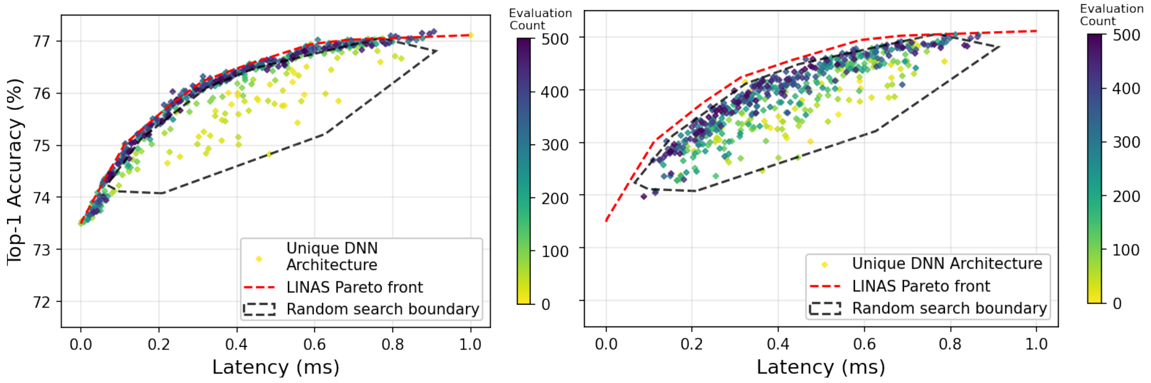


Figure 15: Hypervolume comparison of LINAS, NSGA-II and Random Sampling search methods applied to the quantized model (INT8) of BootstrapNAS ResNet50 super-network. Shaded regions show the standard error for 5 trials with different random seeds.



(a) LINAS Search

(b) NSGA-II Search

Figure 16: Search results in the BootstrapNAS ResNet50 INT8 search space (CLX, batch size = 128) comparing (a): LINAS, and (b): NSGA-II approaches.

H Search Algorithm Details

H.1 Overview

The foundational goal of NAS is to find DNN architectures that are optimal for one or more performance objectives. In the context of weight-sharing super-networks, consider a pre-trained super-network with weights W, a set of sub-network architectural configurations Ω derived from the super-network and m competing objectives $f_1(\omega; W), \ldots, f_m(\omega; W)$ where $\omega \in \Omega$. Each of the sub-network configurations ω is a valid set of parameters used during training of the super-network. For example, a given ω will contain values for each design parameter (e.g., depth and kernel size) used during super-network training. Our system aims to minimize a subset of objectives $S_i \subseteq \{f_1(\omega; W), \ldots, f_m(\omega; W)\}$ to discover the near-optimal sub-network ω_i^*. In other words,

$$\omega_i^* = \underset{\omega \in \Omega}{\operatorname{argmin}}\big(S_i\big) \tag{1}$$

An objective can be negated to transform a minimization objective into a maximization objective (e.g., accuracy is a maximization objective). During optimization, multiple architectures $\omega_i^* \in \Omega$ will be scored in the objective space allowing for the identification of a *Pareto front*, as illustrated in Figure 3c.

H.2 Search Algorithm Settings

In this work we focus on examining random search, multi-objective sequential model-based optimization (SMBO), and multi-objective evolutionary algorithm (MOEA) approaches to the subnetwork search problem. From a hardware-aware standpoint, we evaluate in the multi-objective (a.k.a. bi-objective) setting as we are focused on finding a highly diverse set of near-optimal architectures across the accuracy and latency trade-off (Pareto front) region. However, we note that our framework works with any number of objectives. To test a SMBO algorithm in our framework we employ the multi-objective tree-structured parzen estimator (MOTPE) as proposed by Ozaki et al. (2020). From the Pareto-based MOEA category, the framework supports the popular NSGA-II (Deb, Pratap, et al., 2002) algorithm and a similar approach called AGE-MOEA (Panichella, 2019). For indicator- and decomposition-based MOEAs we support U-NSGA-II (Deb and Sundar, 2006), MOEA/D (Q. Zhang and H. Li, 2007), and CTAEA (K. Li et al., 2019).

For the search algorithm comparison study in Section 3, we evaluate the performance of various evolutionary algorithms, a SMBO multi-objective tree-structured parzen estimator (MOTPE), and a random search using the MobileNetV3 super-network. The evolutionary algorithm settings used for the experiments are shown in Table 4. Evolutionary algorithms that support two or more objectives typically fall in the categories of indicator- or decomposition-based algorithms where the latter often use a predefined set of reference directions on a unit simplex to create objective space partitions. For generating a well-spaced set of reference points from the objective space origin we use the Riesz s-Energy approach (Blank, Deb, et al., 2021). For the MOTPE parameters, we use the recommended settings provided by the authors Akiba et al. (2019) including a prior weight of 1.0 and number of candidate samples used to calculate the expected hypervolume improvement equal to 24.

For the LINAS and NSGA-II experiments across different modalities (Figures 3, 4, 5, 11) we show parameter settings in Table 5. In our ablation studies, cross-over rates between 0.9 and 1.0 performed nearly the same, smaller populations work well with smaller search space sizes, and a mutation rate equal to the inverse of the population size as recommended by (Blank and Deb, 2020) gives the best search performance for NSGA-II. The same settings were used for the LINAS inner-loop predictor search which also often uses NSGA-II in this work. An important note is that NSGA-II and AGE-MOEA are not compatible with three or more objectives and that the other many-objective EA approaches, such as U-NSGA-III would need to be considered in that setting.

Table 4: Evolutionary algorithm parameter settings for the comparison study in Figure 10. Settings generally follow those recommended by Blank and Deb (2020) for each algorithm.

	Evolutionary Algorithm				
	NSGA-II	AGE-MOEA	U-NSGA-III	C-TAEA	MOEA/D
Number of supported objectives	2	2	≥ 2	≥ 2	≥ 2
Population size	50	50	50	-	-
Mutation probability	0.02	0.02	0.02	0.05	-
Crossover probability	0.9	0.9	0.9	1.0	-
Reference direction method	-	-	Riesz s-Energy (20 partitions)	Riesz s-Energy (20 partitions)	Riesz s-Energy (20 partitions)
Number of neighbors	-	-	-	-	20
Neighbor mating probability	-	-	-	-	0.9

Table 5: Experiment settings for the LINAS (with NSGA-II internal loop) and NSGA-II comparison studies in Figures 3, 4, 5, 11. The predictor types apply only to the LINAS setup.

Super-Network (Modality)	Transformer (Machine Translation)	MobileNetV3, ResNet50 (Image Classification)	NCF (Recommendation)
Accuracy Predictor	SVR w/ RBF kernel	Ridge	SVR w/ RBF kernel
Latency Predictor	Ridge	Ridge	SVR w/ Linear kernel
Search Space	10^{15}	10^{19}	10^{7}
Population	50	50	10
Crossover	0.9	0.9	0.1
Mutation	0.02	0.02	0.02
LINAS evaluations (Predictor)	20000	20000	2000