# A COMPILER-AWARE FRAMEWORK OF NETWORK PRUNING SEARCH ACHIEVING BEYOND REAL-TIME MOBILE ACCELERATION

**Yanyu Li** [*1] **Geng Yuan** [*1] **Zhengang Li** [*1] **Wei Niu** [2] **Pu Zhao** [1] **Peiyan Dong** [1] **Yuxuan Cai** [1] **Xuan Shen** [1] **Zheng Zhan** [1] **Zhenglun Kong** [1] **Qing Jin** [1] **Bin Ren** [2] **Yanzhi Wang** [1] **Xue Lin** [1]

## ABSTRACT

With the increasing demand to efficiently deploy DNNs on mobile edge devices, it becomes much more important to reduce unnecessary computation and increase the execution speed. Prior methods towards this goal, including model compression and network architecture search (NAS), are largely performed independently and do not fully consider compiler-level optimization which is a must-do for mobile acceleration. In this work, we propose NPS, a compiler-aware unified network pruning search and the corresponding comprehensive compiler optimizations supporting different DNNs and different pruning schemes, which bridge the gap of weight pruning and NAS. Our framework achieves 6.7ms, 5.9ms, and 3.9ms ImageNet inference times with 77%, 75% (MobileNet-V3 level), and 71% (MobileNet-V2 level) Top-1 accuracy respectively on an off-the-shelf mobile phone, consistently outperforming prior work.

## 1 INTRODUCTION

The growing popularity of mobile AI applications and the demand for real-time Deep Neural Network (DNN) executions raise significant challenges for DNN accelerations. However, the ever-growing size of DNN models causes intensive computation and memory cost, which impedes the deployment on resource limited mobile devices.

DNN weight pruning (Han et al., 2015; He et al., 2018) has been proved as an effective model compression technique that can remove redundant weights of the DNN models, thereby reducing storage and computation costs simultaneously. Existing work mainly focus on *unstructured pruning* scheme (Han et al., 2015) where arbitrary weight can be removed as shown in Fig. 1 (a), and (coarse-grained) *structured pruning* scheme (Zhuang et al., 2018) to eliminate whole filters/channels as shown in Fig. 1 (b). The former results in high accuracy but limited hardware parallelism (and acceleration), while the latter is the opposite. Recent work (Dong et al., 2020; Niu et al., 2020) propose to prune the weights in a more fine-grained manner, which can be classified into block-based and pattern-based pruning as shown in Fig. 1 (c) and (d). This kind of semi-structured pruning preserves higher accuracy while also provides significant speedup with the assist of compiler-level code generation techniques.

Another active research area is the Neural Architecture Search (NAS) (Zoph & Le, 2017), which designs more efficient DNN architectures using automatic searching algorithms. EfficientNet (Tan & Le, 2019) and MobileNetV3 (Howard et al., 2019) are representative lightweight networks obtained by using NAS approaches. Hardware-aware NAS (Cai et al., 2018; Tan et al., 2019) has also been investigated targeting acceleration on actual hardware platforms.

Recently, compiler-assisted DNN inference frameworks (Lane et al., 2015; Xu et al., 2018) have drawn broad attention from both industry and academia. TensorFlow-Lite (TFLite) (Ten), Alibaba Mobile Neural Network (MNN) (Ali), and TVM (Chen et al., 2018) are representative state-of-the-art frameworks that support DNN inference on mobile devices. Recent work PatDNN (Niu et al., 2020) employs a set of compiler-based optimizations to support specific pattern-based sparse DNN models to accelerate the end-to-end inference on mobile devices. However, it still lacks the support for a layer-wise sparse model with various pruning schemes, which significantly limits the versatility of such framework.

Pruning a DNN model for real-time AI applications on mobile devices is a complex task because different types of layers may prefer different types of pruning schemes. At the same time, different layers may show different sensitivities to the pruning ratio. Moreover, even under the similar pruning ratio, different pruning schemes also perform different

[*]Equal contribution [1]Northeastern University [2]College of William and Mary. Correspondence to: Yanyu Li <li.yanyu@northeastern.edu>, Xue Lin <xue.lin@northeastern.edu>.

*Proceedings of the $4^{th}$ MLSys Conference*, San Jose, CA, USA, 2021. 

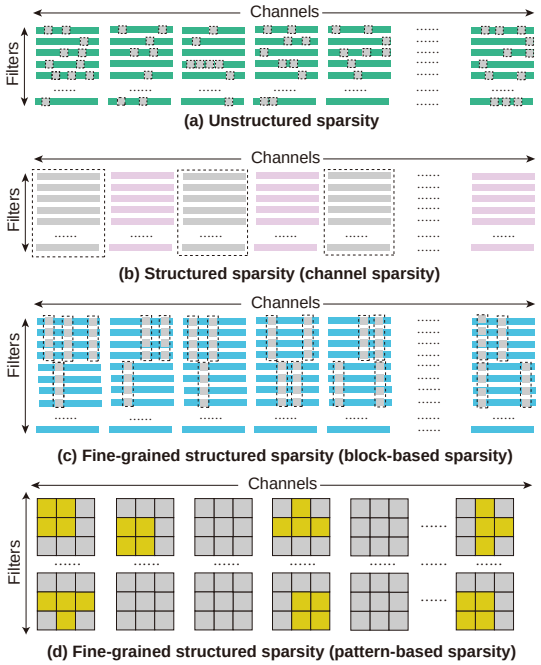


*Figure 1.* Different weight pruning schemes for CONV and FC layers using 4D tensor and 2D matrix representation.

acceleration rates due to computing parallelism. Thus, we bridge the weight pruning technique and NAS methods and propose a einforcement learning (RL)-based network pruning search framework to automatically search the best-suited pruning configurations such as per-layer pruning scheme and pruning ratio. Moreover, we propose multiple compiler optimizations to enable fast code generation and support inference acceleration with per-layer pruning schemes and ratios. We incorporate the compiler optimized model inference latency measured on the target mobile device as a reward in the searching process, making our framework compiler-aware.

Our key contributions include:

- We bridge the gap between network pruning and NAS. We develop a compiler-aware framework of network pruning search, maximizing accuracy while satisfying inference latency constraint.
- We propose comprehensive compiler optimizations supporting different pruning schemes and sparse model inference with per-layer pruning schemes.
- We design a systematic search acceleration strategy, integrating pre-trained starting points, fast accuracy and latency evaluations, and Bayesian optimization.
- Our NPS framework achieves by far the best mobile acceleration: 6.7ms, 5.9ms, and 3.9ms ImageNet inference times with 77.0%, 75%, and 71% Top-1 accuracy, respectively, on an off-the-shelf mobile phone.

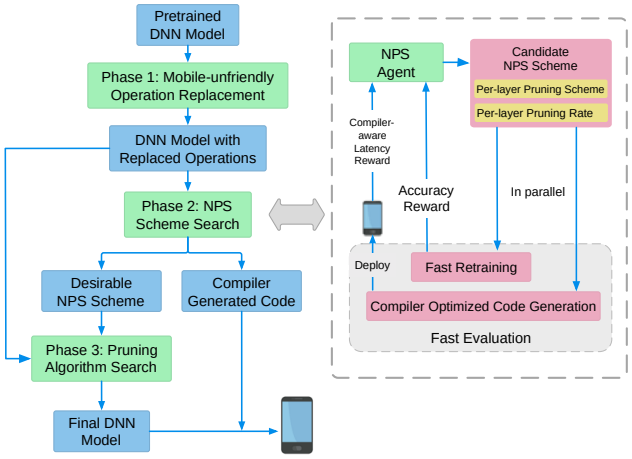


*Figure 2.* Overview of the proposed NPS framework.

## 2 Proposed Unified Network Pruning Search (NPS) Framework

### 2.1 Overview of NPS Framework

Fig. 2 shows the proposed NPS framework. To take advantage of recent NAS results and accelerate the NPS process, we start from a pre-trained DNN model, and go through three phases as shown in the figure.

**Phase 1: Replacement of Mobile-Unfriendly Operations:** Certain operators are inefficient to execute on mobile devices (mobile CPU and GPU). For instance, certain activation functions, such as sigmoid, swish, require exponential computation, and can become latency bottleneck on mobile inference. These unfriendly operations will be replaced by mobile-friendly alternatives such as hard-sigmoid and hard-swish, with negligible effect on accuracy.

**Phase 2: NPS Scheme Search:** This phase uses a RL-based NAS method to generates and evaluates candidate *NPS schemes*, and finally chooses the best-suited one. The search space includes per-layer pruning scheme and per-layer pruning rate. To accelerate such search, we present *a meta-modeling procedure based on RL with Bayesian Optimization* (BO). *A fast evaluation methods* are developed, tailored to NPS framework.

Moreover, we incorporate the overall DNN latency constraint effectively in the reward function of NPS scheme search, ensuring that such constraint can be satisfied at the search outcome. The overall DNN latency is actually measured on the target mobile CPU/GPU based on the candidate NPS scheme currently under evaluation. We rely on actual

measurement instead of per-layer latency modeling as many prior NAS work. This is because our advanced compiler optimizations incorporate a strong layer fusion beyond prior compiler work, which is critical for efficient implementation of super-deep networks, and will make per-layer latency modeling less accurate.

**Phase 3: Pruning Algorithm Search:** We search the most desirable pruning algorithm to perform actual pruning and retrain the remaining weights. The candidate pruning algorithms include magnitude-based ones (Han et al., 2016), ADMM-based algorithm (Zhang et al., 2018), etc.

### 2.2 Fast Evaluation Methods

We develop and adopt multiple tailored acceleration strategies to facilitate fast evaluation in NPS scheme search. To evaluate each generated candidate scheme during search, we use the one-shot magnitude pruning instead of using complex pruning algorithms. And we adopt early stopping strategy, which only retraining pruned model for a few epochs. Because we can distinguish the performance of a candidate NPS scheme by comparing the relative accuracy to other NPS schemes.

Moreover, we overlap the compiler optimization process with the accuracy evaluation process to further accelerate the overall evaluation process. We use compiler code generation and actual on-device latency measurement because of (i) higher accuracy than per-layer latency modeling due to layer fusion mechanism, and (ii) the fast auto-tuning capability of compiler to different mobile devices. Please note that the compiler code generation and latency measurement *do not need the absolute weight values*. Compiler code generation is much faster than DNN training (even a single epoch), and can be performed in parallel with accuracy evaluation (as accurate weight values are not needed). As a result, it will not incur extra time consumption to NPS.

## 3 Compiler Design and Optimizations

Another source of acceleration to achieve real-time inference on mobile devices is the compiler optimizations for generating efficient execution codes. We develop a comprehensive,compiler-based automatic code generation method with multiple optimizations.

**Support for Various Pruning Schemes:** We design a domain specific language (DSL) to represent the DNN model, and a layer-wise representation (LR) is used to describe each DNN layer. This provide us the flexibility for supporting the layer-wise pruning scheme selection. We also design compact weight storage formats for different pruning schemes to improve the data locality.

**Layer Fusion Mechanism:** We incorporate a layer fusion technique to fuse the computation operators in computation graph and effectively reduce the inference latency. Our fusion based on two kinds of properties in the polynomial calculation: *computation laws* (i.e., associative property, commutative property, and distributive property) and *data access patterns*. As a result, we reduce not only the memory consumption of intermediate results, but also the number of operators.

**Auto-tuning for Different Mobile CPU/GPU:** To find the best-suited performance-critical tuning parameters, such as the data placement on GPU memory, matrix tiling sizes, loop unrolling factors, we use auto-tuning approaches as other DNN inference frameworks like TVM. And we incorporate Genetic Algorithm to explore the best configuration automatically and efficiently.

**Compiler-aware Latency:** The latency of a given candidate model is hard to be accurately estimated based on a layer-wise latency model when compiler optimizations are incorporated, especially with layer fusion and auto-tuning. Thus, during the search process, we use real-world compiler optimized latency measured on the real device instead of building a layer-wise latency model. Since the code generation time of our optimized compiler design is much shorter than the accuracy evaluation process, we overlap the code generation and latency measurement with the accuracy evaluation process, hence no extra time cost will be incurred.

*Table 1.* Mobile CPU/GPU Inference latency (ms) comparison with MNN, TVM, and TFLite using dense (unpruned) models. Representative networks (VGG-16, ResNet-18, MobileNet-V2) are evaluated.

| Framework | VGG-16 | ResNet-18 | MobileNet-V2 |
|---|---|---|---|
| TFLite | 429 / 307 | 108 / 49.9 | 55.2 / 24.3 |
| TVM | 251 / 221 | 61.5 / 37.6 | 23.1 / 20.5 |
| MNN | 239 / 141 | 52.4 / 23.7 | 18.6 / 14.5 |
| Ours (dense) | 204 / 103 | 41.1 / 19.8 | 17.4 / 9.3 |
| Ours (sparse) | 37.3 / 18.1 | 20.6 / 9.7 | 9.2 / 4.3 |

**Comparison with Representative DNN Inference Acceleration Frameworks on Mobile Device:** To demonstrate the generality and the superiority of our compiler optimizations, we compared the inference latency of both dense model and sparse model with other representative DNN inference acceleration frameworks including TFLite, TVM, and MNN. And we show the results on widely used benchmark networks including VGG-16, ResNet-18 and MobileNet-V2. Tests are conducted on a Samsung Galaxy S10 smartphone with mobile CPU and mobile GPU respectively. As shown in Table 1, only based on our compiler optimization (without pruning), our results clearly outperforms the representative frameworks on both mobile CPU and mobile GPU. By incorporating our network pruning

search (without causing accuracy loss), the inference latency is further reduced. The pruning rate for VGG-16, ResNet-18, and MobileNet-V2 is 8.2×, 5.3×, and 1.8×, respectively.

## 4 Results and Evaluation

### 4.1 Experimental Setup

We use the image classification task and ImageNet dataset to show the effectiveness of our framework, as in Fig. 3 and 4. We compare our accuracy and latency results with representative DNN inference acceleration frameworks including MNN, PyTorch Mobile, and TFLite. The results are tested on a Samsung Galaxy S20 smartphone using a Qualcomm Snapdragon 865 Octa-core mobile CPU and a Qualcomm Adreno 650 mobile GPU.

For Phase 1, we conduct a fast fine-tuning with 5 training epochs after replacing the mobile-unfriendly operations (only once for the entire NPS process). In Phase 2, 40 Nvidia Titan RTX GPUs are used to conduct the fast accuracy evaluation for candidate NPS schemes concurrently. Since we start from a well-trained model, we retrain 2 epochs for each candidate one-shot pruned model for fast evaluation. For each candidate model, we measure 100 runs of inference on target mobile devices and use the average value as end-to-end latency. Thanks to our fast evaluation and BO, using EfficientNet-B0 as starting point, the overall searching time is 15 days, where Phase 1 only takes 5 epochs, and Phase 3 takes 1.5 days.

*Table 2.* Comparison results of NPS and representative lightweight networks: MobileNet-V1 (Howard et al., 2017), MobileNet-V2 (Sandler et al., 2018), MobileNet-V3 (Howard et al., 2019), NASNet-A (Zoph et al., 2018), AmoebaNet-A (Real et al., 2019), MnasNet-A1 (Tan et al., 2019), ProxylessNas-R (Cai et al., 2018).

| | MACs | Acc. top-1 | Latency (ms) CPU/GPU | Device |
|---|---|---|---|---|
| MobileNet-V1 | 575M | 70.6 | - / - | - |
| MobileNet-V2 | 300M | 72.0 | - / - | - |
| MobileNet-V3 | 227M | 75.2 | - / - | - |
| NAS-Net-A | 564M | 74.0 | 183 / NA | Google Pixel 1 |
| AmoebaNet-A | 555M | 74.5 | 190 / NA | Google Pixel 1 |
| MnasNet-A1 | 312M | 75.2 | 78 / NA | Google Pixel 1 |
| ProxylessNas-R | NA | 74.6 | 78 / NA | Google Pixel 1 |
| NPS (ours) | 290M | 77.0 | 11.8 / 6.7 | Galaxy S20 |
| NPS (ours) | 201M | 75.0 | 9.8 / 5.9 | Galaxy S20 |
| NPS (ours) | 147M | 70.9 | 6.9 / 3.9 | Galaxy S20 |
| NPS (ours) | 98M | 68.3 | 5.6 / 3.3 | Galaxy S20 |

### 4.2 Evaluation Results

First, our compiler optimizations can effectively speed up inference by up to 46% and 141% (on MobileNet-V3) without incorporating NPS compared to the currently best framework MNN on mobile CPU and GPU, respectively. With the highest accuracy (77.0% Top-1), the end-to-end inference time of NPS solution (290M MACs) is only 11.8ms and 6.7ms on mobile CPU and GPU, respectively. With MobileNet-V3 level accuracy (75% Top-1), our inference time (201M MACs) is 9.8ms and 5.9ms. With MobileNet-V2 level accuracy (71% Top-1), the inference time of NPS solution (147M MACs) is 6.9ms and 3.9ms. To the best of our knowledge, this is never accomplished by any existing NAS or weight pruning work. Detailed results can be found in Table 2.

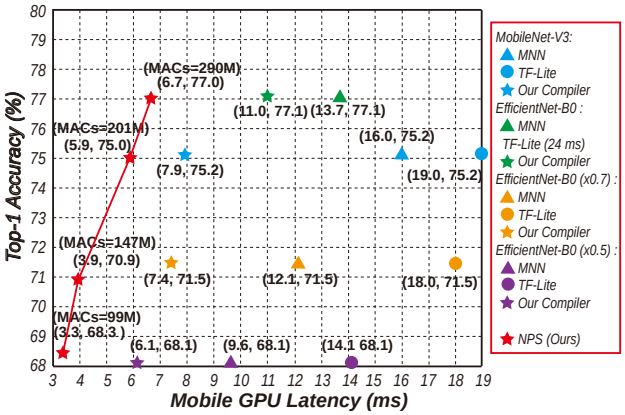


*Figure 3.* Accuracy vs. Latency comparison on mobile GPU.

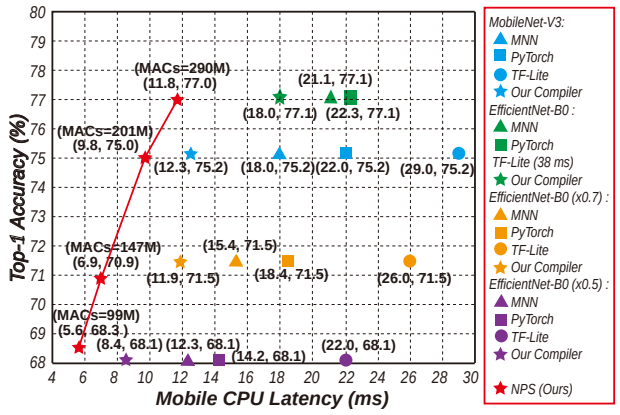


*Figure 4.* Accuracy vs. Latency comparison on mobile CPU.

## 5 Conclusion

In this work, we propose (i) a fine-grained structured pruning applicable to various DNN layers, and (ii) a compiler automatic code generation framework supporting different DNNs and different pruning schemes, which bridge the gap of model compression and NAS. We further propose NPS, a compiler-aware unified network pruning search, and several techniques are used to accelerate the searching process.

## REFERENCES


https://github.com/alibaba/MNN.

https://www.tensorflow.org/mobile/tflite/.

Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Chen, T. et al. Tvm: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594, 2018.

Dong, P., Wang, S., Niu, W., Zhang, C., Lin, S., Li, Z., Gong, Y., Ren, B., Lin, X., Wang, Y., et al. Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition. *arXiv preprint arXiv:2002.11474*, 2020.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems (NeurIPS)*, pp. 1135–1143, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018.

Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

Howard, A. et al. Searching for mobilenetv3. In *ICCV*, 2019.

Lane, N. D. et al. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 283–294. ACM, 2015.

Niu, W., Ma, X., Lin, S., Wang, S., Qian, X., Lin, X., Wang, Y., and Ren, B. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. *arXiv preprint arXiv:2001.00138*, 2020.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Tan, M. et al. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2820–2828, 2019.

Xu, M. et al. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pp. 129–144. ACM, 2018.

Zhang, T., Ye, S., Zhang, Y., Wang, Y., and Fardad, M. Systematic weight pruning of dnns using alternating direction method of multipliers. *arXiv preprint arXiv:1802.05747*, 2018.

Zhuang, Z. et al. Discrimination-aware channel pruning for deep neural networks. In *NeurIPS*, 2018.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 8697–8710, 2018.