A Compiler-aware Framework of Network Pruning Search Achieving Beyond Real-Time Mobile Acceleration

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

Northeastern University & College of William and Mary

Introduction

Deep Learning (DL)

- Huge success in academia, industry, and real life

Large Model Sizes: Challenges

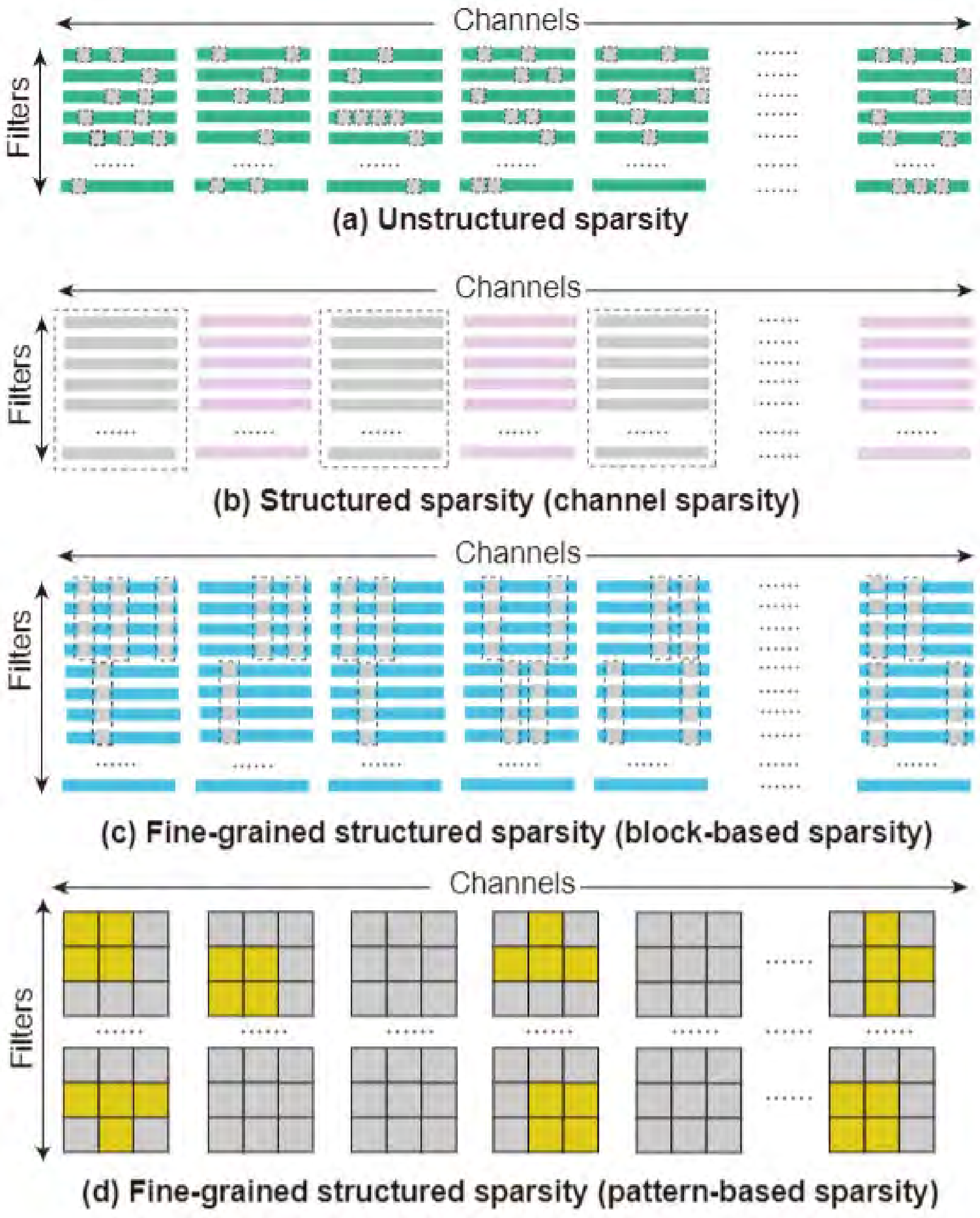
- It is projected that a lot of deep learning systems (the inference phase) will be deployed in edge devices
- The computational complexity makes it hard to deploy large-scale, multi-modal deep learning systems

Background

➢ **Network pruning**

1. Pruning Scheme

2. Pruning Algorithm



Background

- ➢ Neural Architecture Search (NAS)
- Reinforcement Learning (RL) methods
- Evolution methods
- One-shot Training
- Gradient-based methods

… …

- ➢ Mobile DNN Framework
- DNN inference framework
- Compiler-based optimization

Contribution

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.

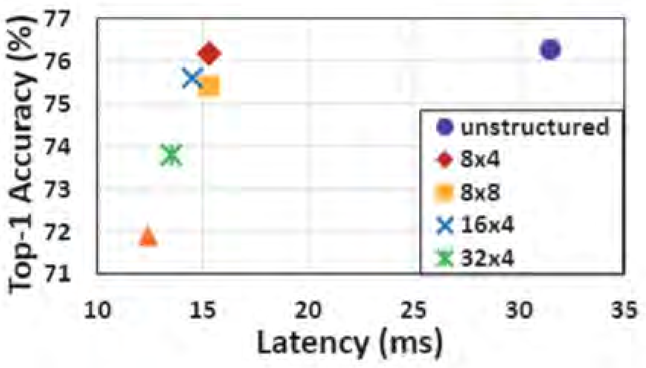
Our NPS framework achieves by far the best mobile acceleration: 6.7ms, 5.9ms, and 3.9ms ImageNet inference times with 77.0%, 75%, and 71% Top-1 accuracy, respectively, on an off-the-shelf mobile phone.

Proposed Fine-grained Structured Pruning

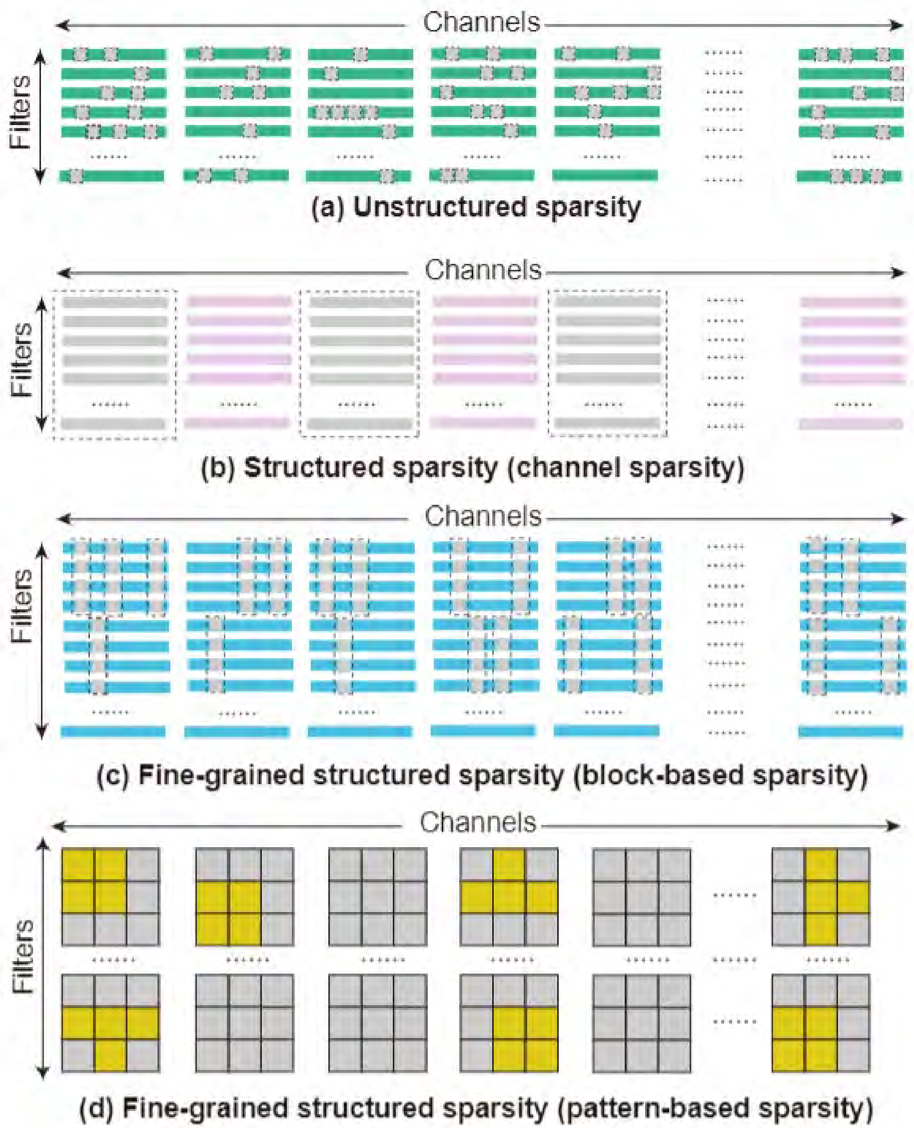
◆ **Proposed New Pruning Schemes:**

1. Block-punched pruning
2. Block-based pruning

◆ **Corresponding Compiler Optimizations**



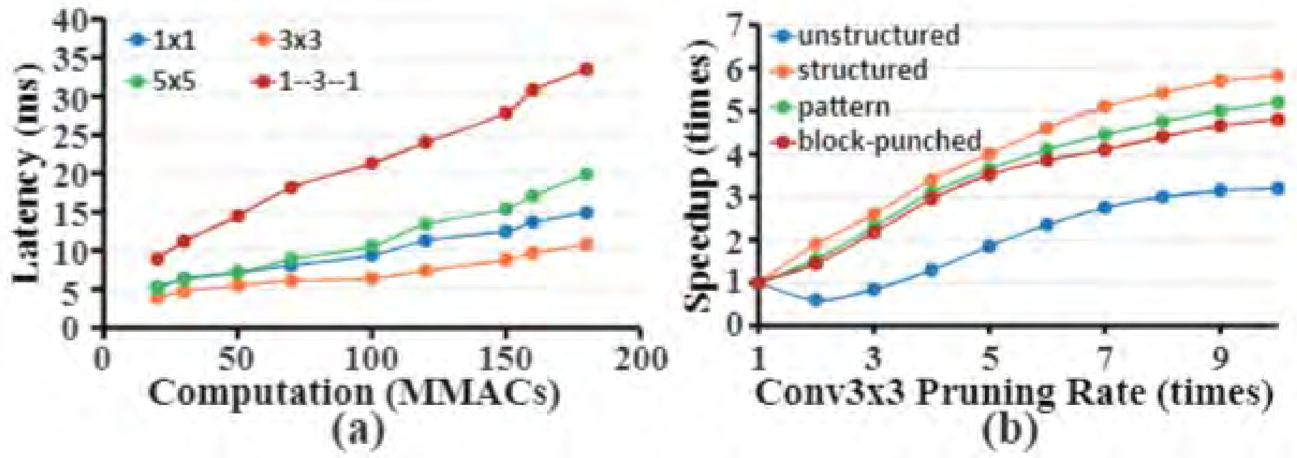
Impact of block size



Proposed Unified Network Pruning and Architecture Search (NPS) Algorithm

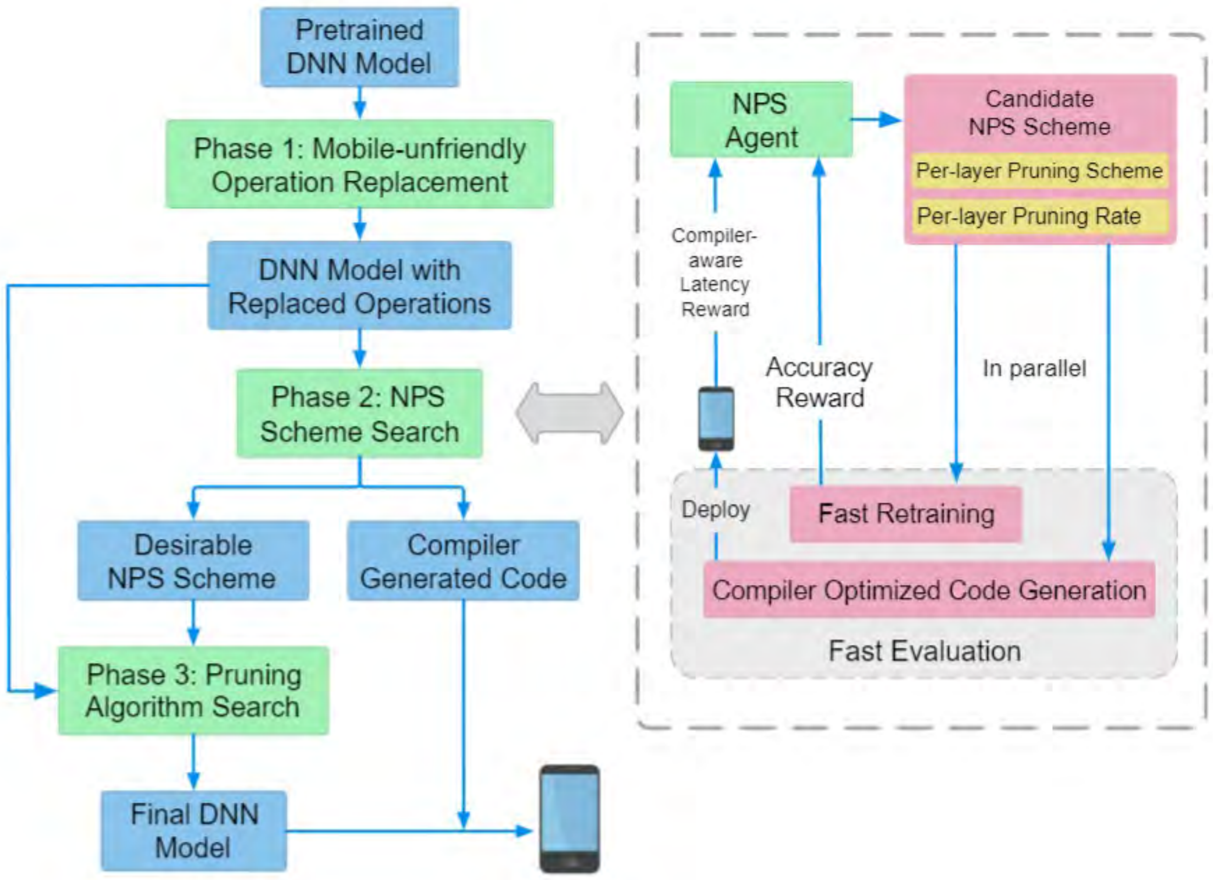
Take into consideration:

- Different Filter Types (Kernel Sizes)
- Different Pruning Schemes



Proposed Unified Network Pruning and Architecture Search (NPS) Algorithm

- Phase 1: Replacement of Mobile-Unfriendly Operations Different Pruning Schemes
- Phase 2: NPS Scheme Search
- Phase 3: Pruning Algorithm Search



Proposed Unified Network Pruning and Architecture Search (NPS) Algorithm

➢Search Space of NPS Scheme

- Per-layer pruning schemes
- Per-layer pruning rate

Pruning scheme	{Filter [49], Pattern-based [31], Block-punched/block-based}
Pruning rate	{ 1×, 2×, 2.5×, 3×, 5×, 7×, 10× }

➢Q-Learning with Baysian Optimization

➢Fast Evaluation Methods

- One-shot Pruning and Early Stopping for Fast Accuracy Evaluation
- Overlapping Compiler Optimization and Accuracy Evaluation

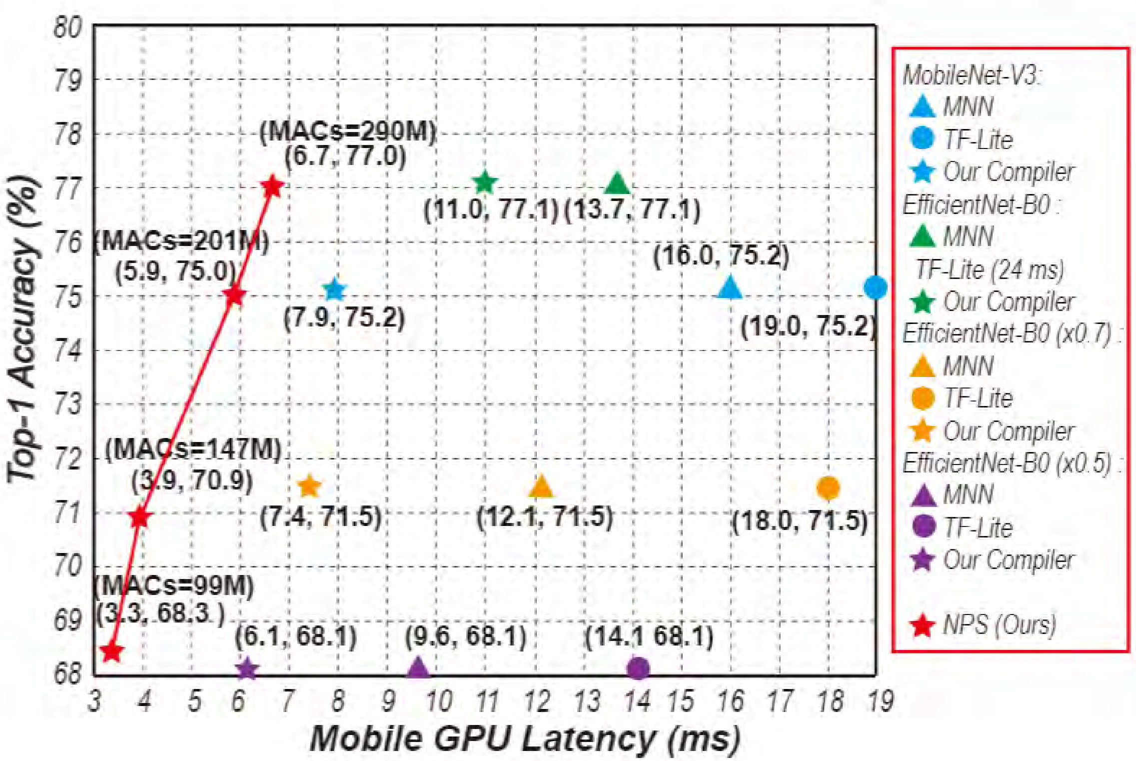
Experiment results

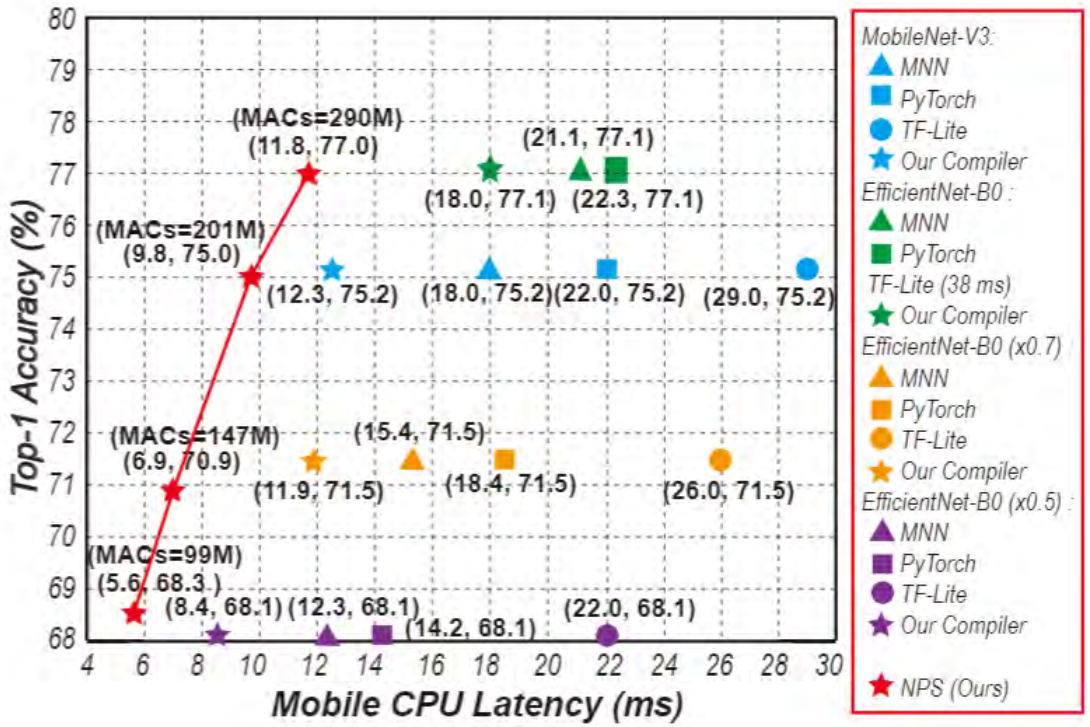
- Our compiler optimization itself can effectively speed up inference by up to 46% and 141% (on MobileNet-V3) without incorporating NPS compared to the currently best framework MNN on mobile CPU and GPU, respectively.

- NPS: 77% -- 6.7ms
- NPS: 75% -- 5.9ms
- NPS: 71% -- 3.9ms

	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

Experiment results





Thank You!