CoCoPIE: Enabling Real-Time AI on Off-the-Shelf Mobile Devices via Compression-Compilation Co-Design

Hui Guan[‡], Shaoshan Liu[†], Xiaolong Ma[◇], Wei Niu[*],
Bin Ren[*], Xipeng Shen[‡], Yanzhi Wang[◇], Pu Zhao[◇]
[‡]North Carolina State University [†]Perceptin Inc.
[◇]Northeastern University [*]William & Mary
Contact: info@cocopie.ai

Abstract

Recent years have witnessed a fast growing interest in creating special hardware for real-time mobile intelligence. It has led to an increasing perception in both research and industry on the indispensability of special AI accelerator for real-time AI on end devices. This article advocates for a cautious treatment to the view. By drawing on a recent real-time AI optimization framework CoCoPIE, it demonstrates that with effective *compression-compiler co-design*, it is possible to enable real-time artificial intelligence on mainstream end devices without special hardware, even for AI tasks as complex as super resolution. The article explains the design of CoCoPIE, and how compression-compiler co-design makes complex AI applications able to run in real-time on off-the-shelf mobile devices that have been previously regarded impossible, helps off-the-shelf mobile devices outperform a number of representative ASIC and FPGA solutions in both energy efficiency and performance, and speeds up DNN pruning by as much as two orders of magnitude.

Many believe that the company who enables real intelligence on end devices (such as mobile devices and IoT devices) will define the future of computing. Racing towards this goal, many companies, whether giant technology firms such as Google, Microsoft, Amazon, Apple and Facebook, or startups spent tens of billions of dollars each year on R&D.

Assuming hardware is the major constraint for enabling real-time mobile intelligence, more and more companies dedicate their main efforts to developing specialized hardware accelerators for machine learning and inference. Billions of dollars have been spent to fuel this intelligent hardware race.

This article challenges the view. By drawing on a recent real-time AI optimization framework CoCoPIE, it maintains that with effective *compression-compiler co-design*, a large potential is yet left untapped in enabling real-time artificial intelligence (AI) on mainstream end devices.

The principle of *compression-compilation co-design* is to design the compression of Deep Learning Models and their compilation to executables in a hand-in-hand manner. This synergistic method can effectively optimize both the size and speed of Deep Learning models, and also can dramatically shorten the tuning time of the compression process, largely reducing the time to the market of AI products. When applied to models running on mainstream end devices, the method can produce real-time experience across a set of AI applications that had been broadly perceived possible only with special AI accelerators.

Foregoing the need for special hardware for real-time AI has some profound implications, thanks to the multi-fold advantages of mainstream processors over special hardware:

- Time to market: Special hardware often takes multiple years before it reaches the market. The creation of the associated compiler and system software further lengthens the process. Applications using such hardware often need to use the special APIs and meet many special constraints (e.g., tiling computations to a certain size), which lengthens the time to market of AI product.
- Cost: Developing a special ASIC processor is costly, and adding them into existing systems incurs extra expenses.
- Technology maturity: Unlike general-purpose processors, special hardware has a much smaller production volume; the technology available for their production is hence usually several generations behind general-purpose processors. Most AI accelerators, for instance, are based on 28 to 65nm CMOS technology, with a transistor density over $10\times$ lower than state-of-art mobile CPU or GPU.
- Speed: As a consequence of the old technology, special processors run much slower than general-purpose processors do.
- Eco-system: General-purpose processors have a well-developed eco-system (debugging tools, optimization tools, security measures), which makes the development of high-quality applications much easier than on special processors.
- Adoption: For all the above reasons, the adoption of a special processor is usually limited to the company that creates it and its few close customers. As a result, an AI application developed for the processor can be adopted by a limited number of devices.

These reasons suggest that whenever mainstream processors can meet the speed and efficiency requirements of an AI application, they should be the preferred device to consider. The current industry however puts much emphasis on special AI hardware development, based on assumed insufficiency of mainstream processors in meeting the real-time requirements. In the rest of this article, we explain why the assumption is largely biased when compression-compilation co-design is used, how the principle can be materialized effectively into a practical framework CoCoPIE, and the implications to the AI industry.

1 Compression-Compilation Co-Design: the Concept

Compression and compilation are the two key steps in fitting a deep learning model on a hardware for efficient executions. Model compression is a common technique for reducing the size and improving the speed of deep learning models. Compression techniques fall into two categories, *pruning* and *quantization*. Pruning removes layers or convolution filters or channels, while quantization reduces the precision of parameters (e.g., from floating-point to short integer). Compilation refers to the process of generating executable codes from a given deep learning model. It, in essence, is a process of mapping the high-level operations in deep learning to the low-level instructions that the underlying hardware supports. The process plays a critical role in optimizing the code for efficient executions.

The principle of compression-compilation co-design is to design the two components for AI in a hand-in-hand manner. The synergy may exhibit at three levels.

(1) Demands/Preferences Level: At this level, the synergy is on taking the preferences or demands of one component into consideration when designing the other component. An example is

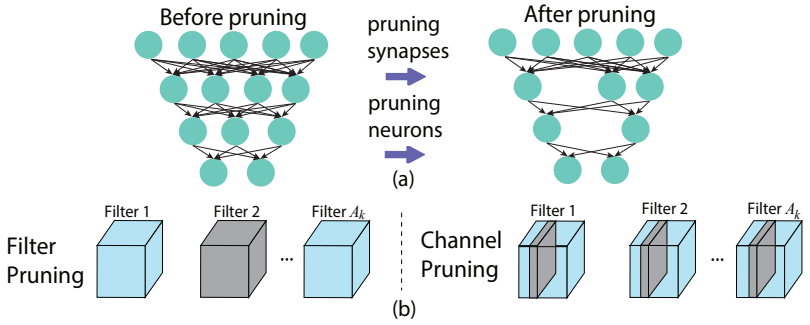


Figure 1: (a) Non-structured weight pruning and (b) two types of structured weight pruning.

the *pattern-based pruning* in Section 2.1, which creates a regular computation pattern amenable for the compilation step to effectively map to vector units or GPU.

(2) Perspective/Insight Level: At this level, the synergy is on taking the perspective or insights in the domain of one component when treating the problems in the domain of the other component. An example is the *composability-based pruning* in Section 2.2, which generalizes the principle of composability or modularity in programming systems into a novel DNN model pruning method to dramatically reduce the needed computations.

(3) Methodology Level: At this level, the synergy is on closely integrating the methodology of the two components together. Section 2.2 illustrates this synergy through a compiler framework that automatically generates code to enable a new way of deep learning pruning, which speeds the process by up to 180X.

All the examples we have mentioned are part of a software framework for Mobile AI named CoCoPIE. We will next give an overview of CoCoPIE based on our previous publications [18, 8], and then use each of its main components to explain the compression-compilation co-design principle and the significant benefits.

2 CoCoPIE

CoCoPIE stands for Compression-Compilation co-design for Performance, Intelligence, and Efficiency. It is a software framework that we have recently put together that holds numerous records on real-time AI on mainstream end devices in both performance and energy efficiency.

CoCoPIE consists of two main components, which both reflect the Compression-Compilation co-design principle. The first component, *CoCo-Gen*, generates efficient DNN execution codes via a synergy of pattern-based DNN pruning and pattern-aware code generation. The second component, *CoCo-Tune*, dramatically shortens the process in identifying the appropriate set of DNN parameters to prune by a composability-based compiler framework. We next explain each of the two components and how compression-compilation co-design makes them possible.

2.1 CoCo-Gen: Pattern-based Pruning and Code Generation

Weight pruning reduces the number of weights in DNN. As shown in Figure 1, prior weight pruning methods fall into two caterogies: (1) general and non-structured pruning where arbitrary weights can be pruned; (2) structured pruning which prunes filters or channels in a way that produces regular and smaller weight matrices. For the better fit of regular structures for GPU/CPU executions, DNNs from regular compression have shown better speeds over those from irregular compression [16, 20, 10], but are subject to more notable accuracy loss [20, 10].

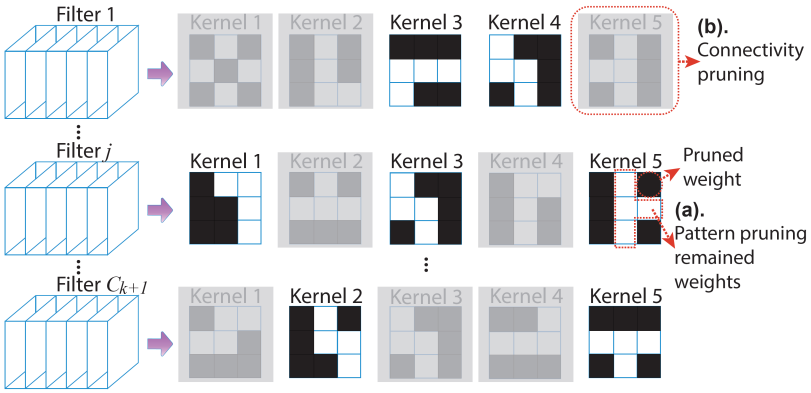


Figure 2: Illustration of (a) kernel pattern pruning on CONV kernels, and (b) connectivity pruning by removing kernels.

2.1.1 Pattern-based Pruning

We introduce a new method, *pattern-based pruning*, which features *fine-grained pruning patterns inside coarse-grained structures*.

Figure 2 illustrates the basic idea of *pattern-based pruning*. For each kernel (in a CONV filter), a fixed number of weights are pruned, and the remaining weights (white cells) form specific "patterns". We define the example in Figure 2 as 4-entry pattern pruning, since every kernel reserves 4 non-zero weights out of the original 3×3 kernel (the most commonly used kernel). It can generalize to other kernel sizes and fully connected layers. Each kernel has the *flexibility* in choosing among a number of pre-defined patterns.

At theory and algorithm levels, such patterns exhibit similarities to the connection structure in human visual systems [15, 13, 12]. At compiler level, the known patterns allow a compiler to *re-order and generate codes* at filter and kernel level such that kernels with the same pattern can be grouped together for consecutive executions, thereby maximizing instruction-level parallelism. At hardware level, 4-entry patterns perfectly fit the SIMD architecture in embedded processors, for both CPUs and GPUs.

The selection of appropriate patterns for a kernel can be achieved via search through an extended ADMM-based framework [15], which can be sped up via a composability-based method in Section 2.2.

The method can be used together with *connectivity pruning*, which *cuts the connections* between certain input and output channels, to achieve even higher weight pruning/acceleration rates.

Figure 3 shows the overview of the internal workflow of CoCo-Gen [18]. After *pattern-based training* performs kernel pattern and connectivity pruning, *execution code generation* performs multiple pattern-based optimizations. Similar to other DNN compilers (e.g., TVM [2]), CoCo-Gen converts DNN models into computational graphs and applies multiple graph-based optimizations. It works on a **fine-grained DNN layerwise representation (LR)** which captures the kernel patterns and tuning-related information. It employs an optimization, **filter kernel reorder**, to address two challenges of pattern-based pruning—heavy control-flow instructions, and thread divergence and load imbalance—by grouping the filters with similar lengths and patterns together. It stores DNN models in a novel form **compressed weight storage**, specifically designed for the kernel patterns and connectivity pruning, yielding a much better compression rate than the conventional compressed sparse row (CSR) format does. It further uses a register-level optimization, **load redundancy elimination**, to maximize memory performance. In sum, allowing compilers to treat

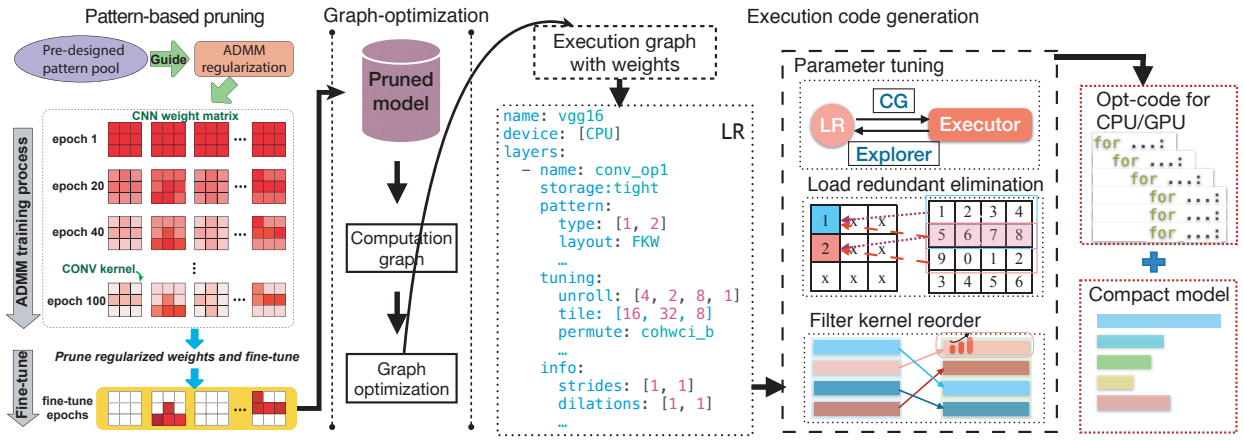


Figure 3: Overview of CoCo-Gen acceleration framework.

pruned kernels as special patterns, the compression-compilation co-design unleashes the power of compiler optimizations for best matching DNN models with underlying hardware (see [18] for more details).

2.2 CoCo-Tune: A Compiler Framework for Fast Pruning

Finding out what is the best set of filters or connectivities to prune—such as the pattern-based pruning in the previous section—can be very time consuming. For a DNN with W filters, the entire configuration space of pruned network can be as large as $2^{|W|}$, even if only filter pruning is considered (adding pattern variations would worsen the complexity further). It often takes hours to evaluate just one configuration (i.e., training the pruned network and then testing it).

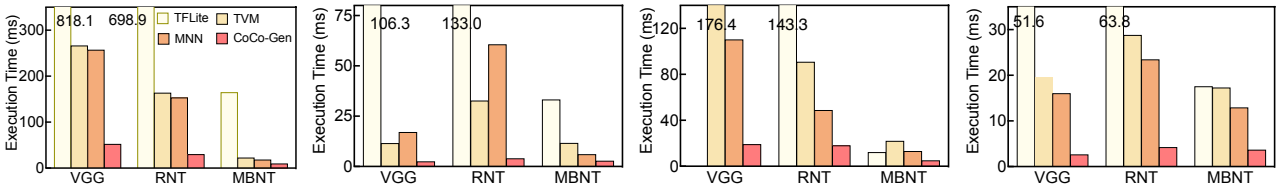
CoCo-Tune is a compiler-based framework that shortens the time by up to 180X. Its inspiration comes from software engineering. More specifically, it explores *composability*, a property (fundamental in software engineering) that we discovered in the training of a collection of pruned CNN models. The basic observation is that two CNN networks in the promising subspace often differ in only some layers, and the training results of the common layers can be reused across networks to save some training time. More generally, CoCo-Tune views the networks to search as compositions of a set of building blocks (a *block* is a sequence of CNN layers). It pre-trains (some of) these building blocks and then assembles them into the to-be-explored networks.

To identify the best set of building blocks to pre-train to maximize the benefits, it uses a novel algorithm, which represents all layers of all to-be-explored networks as a sequence of symbols, and uses a hierarchical compression algorithm Sequitur [17] to produce a context free grammar (CFG) and uses it to quickly find out the most reusable building blocks.

We integrate the techniques into a compiler-based framework, CoCo-Tune, which, for an arbitrary CNN (in Caffe Prototxt format) and other inputs, automatically generates TensorFlow code to build Teacher-Student learning structures to materialize composability-based CNN pruning (see [8] for more details).

2.3 Evaluation and Demos

Results on DNNs: We evaluate CoCo-Gen on a Samsung Galaxy S10 cell phone with the latest Qualcomm Snapdragon 855 mobile platform that consists of a Qualcomm Kryo 485 Octa-core CPU and a Qualcomm Adreno 640 GPU. Six representative DNNs are used in this evaluation, VGG-16 (VGG), ResNet-50 (RNT), and MobileNet-V2 (MBNT) trained on two datasets, ImageNet



(a) ImageNet-CPU (b) CIFAR-10-CPU (c) ImageNet-GPU (d) CIFAR-10-GPU

Figure 4: Performance comparison: x-axis: different trained DNN models; y-axis: average DNN inference execution time on a single input.

Table 1: DNNs characteristics (under kernel pattern and connectivity pruning)

Name	Network	Dataset	Layers	Conv	Patterns	Accu(%)	Accu Loss (%)
VGG	VGG-16	ImageNet	16	13	8	91.6	0.1
		CIFAR-10	16	13	8	93.9	-0.4
RNT	ResNet-50	ImageNet	50	49	8	92.5	0.2
		CIFAR-10	50	49	8	95.6	-1.0
MBNT	MobileNet -V2	ImageNet	53	52	8	90.3	0.0
		CIFAR-10	54	53	8	94.6	-0.1

* Accu: accuracies of ImageNet top-5, CIFAR top-1; negative Accuracy Loss means accuracy improvement.

and CIFAR-10, respectively. Table 1 characterizes these DNNs and lists the number of pruning patterns and the loss of prediction accuracy caused by our pruning. Figure 4 shows the CPU and GPU performance of CoCo-Gen compared to TFLite [6], TVM [2], and MNN [1]. CoCo-Gen outperforms all other frameworks for all cases. On CPU, CoCo-Gen achieves $12\times$ to $44.5\times$ speedup over TFLite, $2.3\times$ to $8.1\times$ over TVM, and $1.9\times$ to $15.5\times$ over MNN, respectively. On GPU, CoCo-Gen achieves $2.5\times$ to $20\times$, $4.1\times$ to $11.4\times$, and $2.5\times$ to $6.2\times$ speedup over TFLite, TVM, and MNN, respectively[1]. For the largest DNN (VGG) and largest data set (ImageNet), CoCo-Gen completes CONV layers on a single input within 18.9 ms on GPU, meeting the real-time requirement (usually 30 frames/sec, i.e., 33 ms/frame).

In terms of energy consumption, CoCo-Gen is $8.6\times$ less than TVM. The power consumption rate of the entire mobile device is 4.1W, slightly higher than that of TVM executions, 3.8W (tested by Qualcomm Trepn power profiler). But its $9.3\times$ less execution time leads to the large savings in energy.

The results also consistently outperform a number of ASIC and FPGA solutions in both performance and energy efficiency. Figure 5 demonstrates (i) the comparison results on performance and energy efficiency with special ASIC hardware including Google's cloud TPU-V2 and edge TPU [7], NVIDIA Jetson AGX Xavier, Cambricon MLU-100, Eyeriss [3], etc., and (ii) comparison results on accuracy and energy efficiency with the FPGA solution ESE [9] (FPGA 2017 Best Paper Award) from DeePhi. The comparisons are on the same network models, and weight quantization is not used in CoCo-Gen solution (Eyeriss and ESE use 12-bit fixed-point quantizations).

The better results of CoCo-Gen come from three reasons: (i) the compression-compilation co-design more effectively matches models with hardware; (ii) smartphone chips are built with the most advanced technology (e.g., 7nm, 11nm technology), while FPGA/ASIC solutions are based

[1]TFLite does not support executing VGG on ImageNet data set on GPU due to its too large memory footprint.

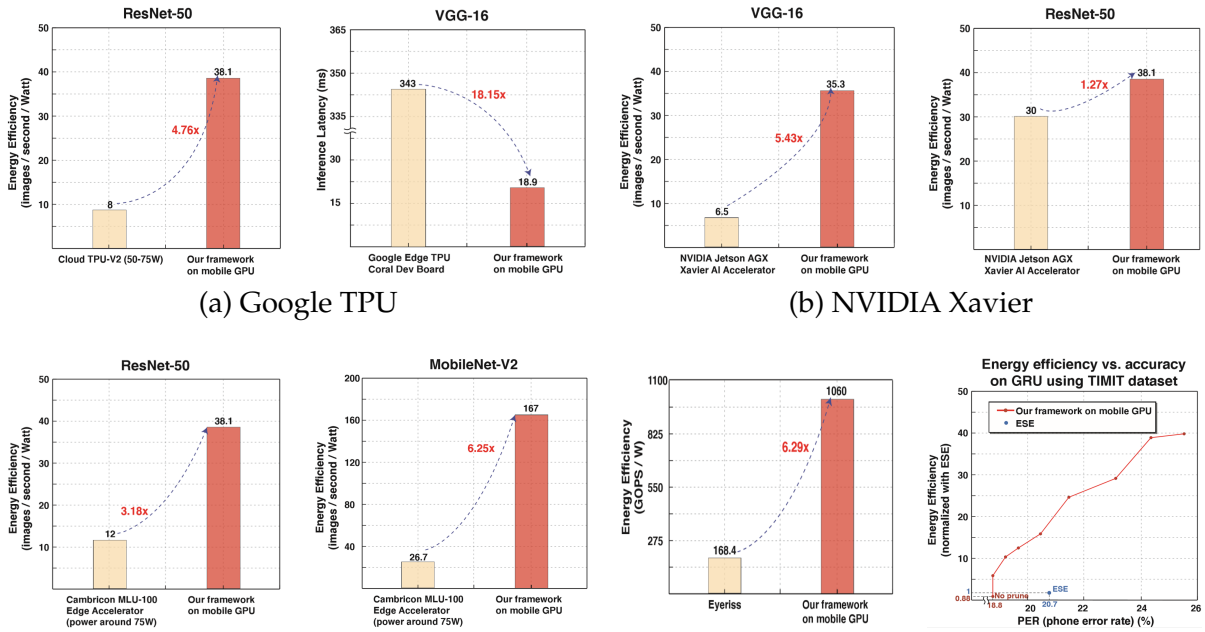


Figure 5: Comparison with existing ASIC and FPGA solutions. (a) Comparison of energy efficiency and inference latency with Google cloud TPU. (b) Comparison of energy efficiency with NVIDIA Jetson AGX Xavier. (c) Comparison of energy efficiency with Cambricon MLU-100. (d) Comparison of energy efficiency with Eyeriss. (e) Comparison of energy efficiency with FPGA solution ESE.

on older and less energy-efficient 28nm or 40nm technologies. (iii) current ASIC/FPGA solutions are often optimized for a specific DNN type/size (e.g., edge TPU for small-scale DNNs, Cambricon MLU-100 for large-scale DNNs), while CoCo-Gen, as a software method, can better adapt to the networks.

Real Application Demos: We also demonstrate the efficacy of CoCo-Gen through three interesting and key DNN applications, style transfer [5], DNN coloring [11], and super resolution [4]. The style transfer model is based on a generative network [23] trained on Microsoft COCO [14]. DNN coloring uses the Places scene [24] dataset to train a novel architecture that can jointly extract and fuse global and local features to perform the final colorization. The super resolution model mainly utilizes residual blocks with wider activation and linear low-rank convolution [21] trained on the DIV2K [19] dataset. With structured pruning and compiler optimization, we implement the models on a Samsung Galaxy S10 mobile phone. We demonstrate that our implementations are able to achieve real-time inference on off-the-shelf mobile device with video demos.

Figure 6 shows sample inputs and outputs of the three applications. According to our study [18], *CoCo-Gen* on unpruned models can already outperform existing DNN frameworks (e.g., TVM) in speed thanks to its advanced optimizations (e.g., operator replacement, SIMD optimizations, etc.). When combined with pattern-based pruning, CoCo-Gen produces $4.2\times$, $3.6\times$, and $3.7\times$ extra speedups on style transfer, coloring and super resolution, respectively. An inference in all the applications can complete within 75 ms, demonstrating the promise of real-time performance of complex DNN applications on the off-the-shelf devices. (Please see video demos

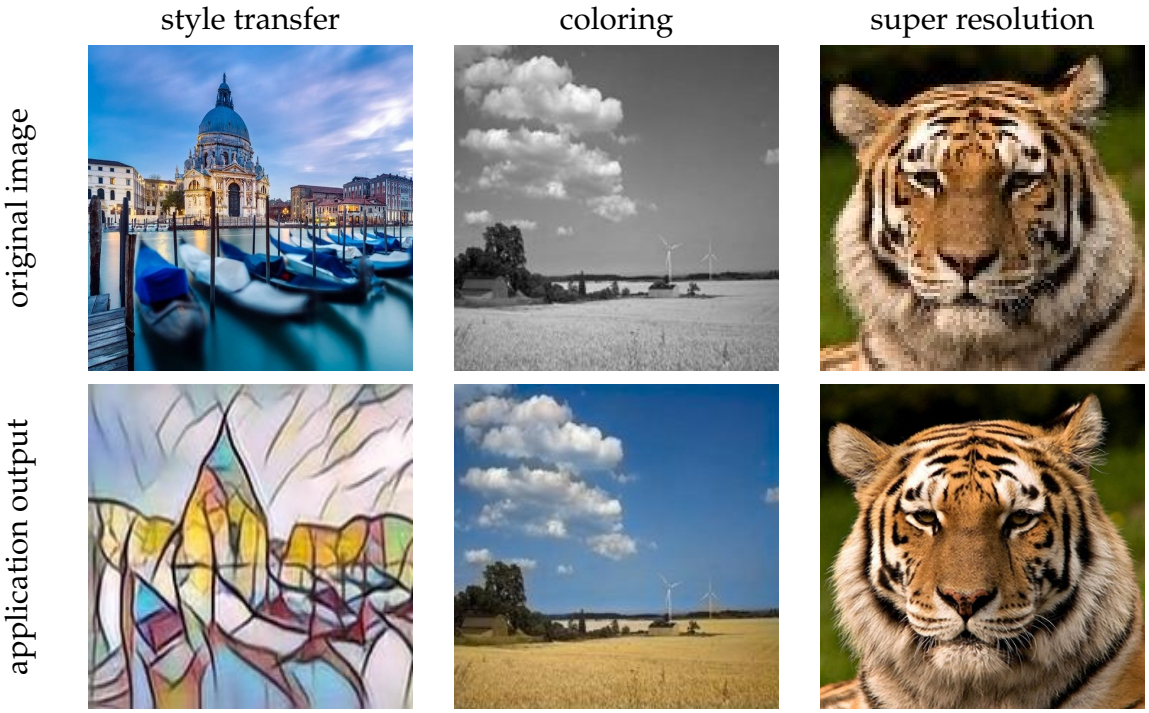


Figure 6: Examples of style transfer, coloring, and super resolution implemented on our mobile device.

of the applications at CoCoPIE YouTube channel[2].

3 Conclusions and Future Work

This article has introduced the concept of *compression-compilation co-design* and how it is materialized into a software framework CoCoPIE for real-time AI on mobile devices. The promising progress opens up many potential directions for future development. We briefly discuss two of them.

The first is to expand the scope of the co-design based optimizations. So far, the principle of compression-compilation co-design has been focused on DNN models. Besides DNN, a real-world AI application often includes a lot of other parts, such as data collection, data preprocessing, the use of the DNN prediction in follow-up operations, and so on. Even though DNN may play an important role in the overall application, its optimizations may not be sufficient for the entire application to meet users' needs. So an important direction is on how to generalize the co-design principle into holistic optimizations to the entire AI-based applications.

The second is to increase the applicability of the co-design based optimizations. This direction relates with privacy and security. As they are two important factors in many AI model constructions and deployments, how to integrate them into the co-design process is worth pursuing. For instance, typically model pruning requires access to both the models and the training datasets, but there are scenarios where datasets may not be accessible to the model optimizer due to either privacy policies or artificial boundaries among corporations. Effective ways to circumvent these roadblocks could expand the applicability of the optimizations [22]. This direction also relates with the way that the optimization framework takes to deliver its service (e.g., standalone

[2] `http://www.youtube.com/channel/UCCKVDtg2eheRTEuqIJ5cD8A/`.)

software versus cloud-based service).

In summary, the research reported in this article has provided strong evidences of the promise of the co-design principle. They indicate that it is possible to instill AI directly on existing commodity computing devices while offering even higher speeds and better energy efficiency than special AI accelerating hardware does. The results open new opportunities for democratizing AI capability on end devices, while cautioning against the broad perception on the indispensability of special AI hardware for real-time AI on end devices. We believe that these results will prompt the industry to reexamine the directions and strategies on the pursuit of mobile AI.

References Cited

[1] Alibaba. Mnn, 2019.

[2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, 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)*, pages 578–594, 2018.

[3] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, pages 262–263, 2016.

[4] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *European conference on computer vision*, pages 184–199. Springer, 2014.

[5] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[6] Google. Tensorflow lite, 2019.

[7] Google Cloud TPU. Google cloud tpu. `https://cloud.google.com/tpu/`, 2017.

[8] Hui Guan, Xipeng Shen, and Seung-Hwan Lim. Wootz: A compiler-based framework for fast cnn pruning via composability. In *Proceedings of the Programming Language Design and Implementation (PLDI)*, 2019.

[9] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William J. Dally. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *FPGA*, pages 75–84, 2017.

[10] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 1398–1406. IEEE, 2017.

[11] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be color! joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Trans. Graph.*, 35(4), July 2016.

[12] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016.

[13] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017.

[14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[15] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. *AAAI*, 2020.

[16] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*, 2017.

[17] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res.(JAIR)*, 7:67–82, 1997.

[18] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. *ASPLOS*, 2020.

[19] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, and Lei Zhang. Ntire 2017 challenge on single image super-resolution: Methods and results. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 114–125, 2017.

[20] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[21] Jiahui Yu, Yuchen Fan, Jianchao Yang, Ning Xu, Zhaowen Wang, Xinchao Wang, and Thomas Huang. Wide activation for efficient and accurate image super-resolution. *arXiv preprint arXiv:1808.08718*, 2018.

[22] Zheng Zhan, Yifan Gong, Zhengang Li, Pu Zhao, Xiaolong Ma, Wei Niu, Xiaolin Xu, Bin Ren, Yanzhi Wang, and Xue Lin. Priv: A privacy-preserving deep neural network model compression framework. *arXiv preprint*, 2020.

[23] Hang Zhang and Kristin Dana. Multi-style generative network for real-time transfer. *arXiv preprint arXiv:1703.06953*, 2017.

[24] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*, pages 487–495, 2014.