Appendix

A Misleading Weight Magnitude

Pruning is often decided by the assumption that small weight magnitudes are less important. There are many pruning works [31,30] based on weight magnitude and certain variants [57,27] using sophisticated regularized training techniques to regularize the weight magnitude for pruning optimizations.

Measuring the weight importance based on their magnitude could be misleading. We take the indicator approach introduced in [57] as baseline, where channel importance is measured by the magnitude of γ in batch normalization layer, as it is a scaling multiplier along channel dimension. With the indicators, we can prune the model with the following methods: (i) We can simply decide pruning in one-shot by pruning channels with smaller magnitudes. (ii) We update the policy iteratively during regularized training as a naive improvement of (i). (iii) To reflect importance shifting during pruning and overcome the weakness of (ii), some work [57,27] equally penalize all indicators to be close to zero.

We observe that (ii) suffers from significant accuracy loss since layers with smaller magnitudes receive more penalty, and are not recoverable. As a result, layers pruned more at the initial will be pruned more and more, leading to a non-recoverable pruning with certain accuracy degradation. In (iii), equal penalization zeros out overmuch information and inevitably destroys the model accuracy. Our method with a mask layer, which prunes weights based on separate variables (as indicators) instead of weight magnitudes, can achieve a higher accuracy than the above (i), (ii), and (iii).

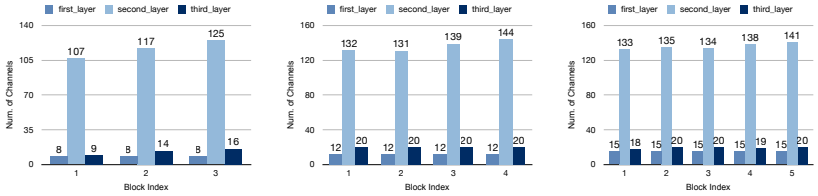
Our mask layer does not depend on the weight magnitude since our pruning indicators are decoupled from model parameters and can be directly trained in binary format.

B Speed Model Performance

The trained speed model is accurate in predicting the speed of different layer width configurations in the WDSR block with 2.95% error. To demonstrate the

predicted (ms)	real (ms)	Δ	%
32.75	33.13	0.38	1.16%
44.64	44.08	-0.56	1.26%
59.28	62.28	2.99	4.81%
61.31	60.46	-0.85	1.41%
87.12	86.83	-0.29	0.34%
120.12	119.44	-0.68	0.57%
137.45	131.39	-6.05	4.61%
173.84	168.87	-4.97	2.94%
230.40	220.28	-10.12	4.59%

Table A1: Comparison between predicted inference time and real inference time on Samsung Galaxy S21 GPU.



(a) $v_t = 40$ms (b) $v_t = 70$ms (c) $v_t = 100$ms

Fig. A1: Searched model architectures for a $v_t = 40$ms, b $v_t = 70$ms and c $v_t = 100$ms for performing 720p $\times 2$ SR task

	method	PSNR				FPS
		Set5	Set14	B100	Urban100	
$\times 2$	CARN-M [7]	37.53	33.26	31.92	31.23	0.4
	FSRCNN [21]	37.00	32.63	31.53	29.88	3.9
	Ours (100ms)	37.64	33.16	31.91	31.08	6.8
$\times 4$	CARN-M [7]	31.92	28.42	27.44	25.62	0.9
	FSRCNN [21]	30.71	27.59	26.98	24.62	11.3
	Ours (70ms)	31.88	28.43	27.46	25.69	12.4

Table A2: Comparison of different SR methods implemented with MNN on mobile CPU.

performance of the trained speed model on general configurations, we further evaluate the trained speed model with randomly generated configurations. We show the prediction/real speed of 9 random configurations in Table A1. The largest error is 4.81% among the 9 examples and 6 of the examples have an error less than 3%. The accurate prediction of the speed model provides the foundations for our search framework to derive real-time SR models with high SR performance.

C Compiler Optimizations

Compiler optimization can support various block width configurations, with the following components.

Fusion. There are multiple kinds of layer operators with various computation patterns in the DNN models. Based on different combinations of computation patterns, we can adopt layer fusion to fuse the computation operators in computation graph. For example, a combination of CONV layer/Depthwise CONV layer and its following BatchNorm layer can be fused into one single layer to reduce the data movement and access. With layer fusion, we can reduce the layer number, and save the parameters and the intermediate computations of fused layers. We classify the existing operations in the SR model into several groups based on the mapping between the input and output, and develop rules for different combinations of the groups in a more aggressive fusion manner.

scale	target (ms)	Params (K)	MACs (G)	speed (ms)	PSNR				SSIM			
					Set5	Set14	bsds100	urban100	Set5	Set14	bsds100	urban100
×2	100	91	21.0	96.34	37.68	33.21	31.91	31.24	0.9586	0.9138	0.8963	0.9189
	70	69	16.0	69.01	37.59	33.12	31.84	31.05	0.9583	0.9129	0.8954	0.9169
	40	37	8.5	32.51	37.34	32.91	31.67	30.54	0.9574	0.9111	0.8933	0.911

Table A3: Searched SR model performances on DSP for implementing 720p resolution

Parameter Auto-Tuning. During the compiler optimization, there are many parameters related to the computation, such as data placement on GPU memory, loop unrolling factors, matrix tiling sizes, etc. To find the best configuration of the parameters, an auto-tuning process is adopted. Specifically, we use a genetic algorithm to explore the parameter space. Besides, the explore efficiency can be improved by increasing the population number in each generation to improve the exploration parallelism.

D Searched Architecture

In Figure A1, we show the searched model architectures of ×2 SR task for 720p resolution on Samsung Galaxy S21 GPU. The horizontal axis indicates the SR block index in the model, and the vertical axis indicates the number of channels in each layer within the block. Different colors denote different layers inside the corresponding SR block. We can observe that deeper blocks usually need more channels.

E Comparison with Other Frameworks

To demonstrate the effectiveness of our compiler optimizations, we implement CARN-M [7], FSRCNN [21], and our searched model with the open-source MNN framework on mobile CPU. As shown in Table A2, our model can achieve higher FPS and PSNR than the baseline models.

F Comparison with Other Compiler

The compilation of [37] released 5 years ago is out-of-date, not matching current competitors, and does not support the PIXEL-SHUFFLE operator in WDSR. Our compiler method is more advanced. To make a fair comparison, we test the speed of VGG-16 based on [37] and our compilation method. VGG-16 with [37] needs 644ms on Galaxy S7, while our compiler on VGG-16 only has 27ms latency on Galaxy S21. Our method performs better since the GPU difference can not lead to 24× speedup.

G Performance on DSP

The proposed framework is general and can achieve real-time inference of implementing 720p resolution not only on the GPU of mobile platforms. To demonstrate this, we further provide the implementation on the digital signal processor (DSP) of a mobile devices (Samsung Galaxy S21). To obtain the models targeting for the mobile DSP, a new speed model needs to be trained with latency data collected on the mobile DSP. The performance of the searched models is shown in Table A3. The latency threshold v_T is set to different values including 100ms, 70ms, and 40ms, for a comprehensive study. From the results we can see that we could achieve fast inference or even real-time SR inference with high image quality on the mobile DSP. We highlight that can be easily extended to other platforms and devices with the corresponding the latency dataset.