Supplementary Material

1. Winograd transformations for convolution

1.1. Standard Winograd transforms

Following [12], given a set of polynomial points (f_i, g_i), the Vandermonde matrix $V_{a\times b}$ is constructed as below,

$$\begin{bmatrix} f_0^0 g_0^{b-1} & f_0^1 g_0^{b-2} & \cdots & f_0^{b-1} g_0^0 \\ f_1^0 g_1^{b-1} & f_1^1 g_1^{b-2} & \cdots & f_1^{b-1} g_1^0 \\ \vdots & \vdots & \ddots & \vdots \\ f_{a-1}^0 g_{a-1}^{b-1} & f_{a-1}^1 g_{a-1}^{b-2} & \cdots & f_{a-1}^{b-1} g_{a-1}^0 \end{bmatrix} \tag{1}$$

For standard Winograd convolution, we adopt the widely used polynomial points and scaling factors, as mentioned in [2, 4, 6, 12]. Specifically, for $F(4,3)$ the polynomial points, scaling factors, Vandermonde and transformation matrices are the following,

$$\begin{aligned} (f_i, g_i) =& [(0,1),(1,1),(-1,1),\\ & (2,1),(-2,1),(1,0)] \\ S_A =& [1,1,1,1,1,1] \\ S_B =& [4,-6,-6,24,24,1] \\ S_G =& [1/4,-1/6,-1/6,1/24,1/24,1] \end{aligned} \tag{2}$$

$$V_{6\times 4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 2 & 4 & 8 \\ 1 & -2 & 4 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$V_{6\times 6}^{-T} = \begin{bmatrix} 1 & 0 & -5/4 & 0 & 1/4 & 0 \\ 0 & 2/3 & 2/3 & -1/6 & -1/6 & 0 \\ 0 & -2/3 & 2/3 & 1/6 & -1/6 & 0 \\ 0 & -1/12 & -1/24 & 1/12 & 1/24 & 0 \\ 0 & 1/12 & -1/24 & -1/12 & 1/24 & 0 \\ 0 & 4 & 0 & -5 & 0 & 1 \end{bmatrix} \tag{4}$$

$$V_{6\times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 1 & -2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$\begin{aligned} A^T &= V_{6\times 4}^T diag(S_A) \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 0 \\ 0 & 1 & 1 & 4 & 4 & 0 \\ 0 & 1 & -1 & 8 & -8 & 1 \end{bmatrix} \end{aligned} \tag{6}$$

$$\begin{aligned} B^T &= diag(S_B) V_{6\times 6}^{-T} \\ &= \begin{bmatrix} 4 & 0 & -5 & 0 & 1 & 0 \\ 0 & -4 & -4 & 1 & 1 & 0 \\ 0 & 4 & -4 & -1 & 1 & 0 \\ 0 & -2 & -1 & 2 & 1 & 0 \\ 0 & 2 & -1 & -2 & 1 & 0 \\ 0 & 4 & 0 & -5 & 0 & 1 \end{bmatrix} \end{aligned} \tag{7}$$

$$\begin{aligned} G &= diag(S_G) V_{6\times 3} \\ &= \begin{bmatrix} 1/4 & 0 & 0 \\ -1/6 & -1/6 & -1/6 \\ -1/6 & 1/6 & -1/6 \\ 1/24 & 1/12 & 1/6 \\ 1/24 & -1/12 & 1/6 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{8}$$

For $F(6,3)$, the polynomial points, scaling factors, Vandermonde and transformation matrices are the following,

$$\begin{aligned} (f_i, g_i) =& [(0,1),(1,1),(-1,1),(2,1),\\ & (-2,1),(1/2,1),(-1/2,1),(1,0)] \\ S_A =& [1,1,1,1,1,1,1,1] \\ S_B =& [1,-9/2,-9/2,90,90,45/32,45/32,1] \\ S_G =& [1,-2/9,-2/9,1/90,1/90,32/45,32/45,1] \end{aligned} \tag{9}$$

$$V_{8\times 6} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 2 & 4 & 8 & 16 & 32 \\ 1 & -2 & 4 & -8 & 16 & -32 \\ 1 & 1/2 & 1/4 & 1/8 & 1/16 & 1/32 \\ 1 & -1/2 & 1/4 & -1/8 & 1/16 & -1/32 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

$$V_{8\times 8}^{-T} = \begin{bmatrix} 1 & 0 & -21/4 & 0 & 21/4 & 0 & -1 & 0 \\ 0 & -2/9 & -2/9 & 17/18 & 17/18 & -2/9 & -2/9 & 0 \\ 0 & 2/9 & -2/9 & -17/18 & -17/18 & 2/9 & -2/9 & 0 \\ 0 & 1/180 & 1/360 & -1/36 & -1/72 & 1/45 & 1/90 & 0 \\ 0 & -1/180 & 1/360 & 1/36 & -1/72 & -1/45 & 1/90 & 0 \\ 0 & 64/45 & 128/45 & -16/9 & -32/9 & 16/45 & 32/45 & 0 \\ 0 & -64/45 & 128/45 & 16/9 & -32/9 & -16/45 & 32/45 & 0 \\ 0 & -1/4 & 0 & 21/4 & 0 & -21/4 & 0 & 1 \end{bmatrix} \tag{11}$$

$$V_{8\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 1 & -2 & 4 \\ 1 & 1/2 & 1/4 \\ 1 & -1/2 & 1/4 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

$$A^T = V_{8\times6}^T diag(S_A) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 1/2 & -1/2 & 0 \\ 0 & 1 & 1 & 4 & 4 & 1/4 & 1/4 & 0 \\ 0 & 1 & -1 & 8 & -8 & 1/8 & -1/8 & 0 \\ 0 & 1 & 1 & 16 & 16 & 1/16 & 1/16 & 0 \\ 0 & 1 & -1 & 32 & -32 & 1/32 & -1/32 & 0 \end{bmatrix} \tag{13}$$

$$B^T = diag(S_B) V_{8\times8}^{-T} = \begin{bmatrix} 4 & 0 & -21 & 0 & 21 & 0 & -4 & 0 \\ 0 & 4 & 4 & -17 & -17 & 4 & 4 & 0 \\ 0 & -4 & 4 & 17 & -17 & -4 & 4 & 0 \\ 0 & 2 & 1 & -10 & -5 & 8 & 4 & 0 \\ 0 & -2 & 1 & 10 & -5 & -8 & 4 & 0 \\ 0 & 8 & 16 & -10 & -20 & 2 & 4 & 0 \\ 0 & -8 & 16 & 10 & -20 & -2 & 4 & 0 \\ 0 & -4 & 0 & 21 & 0 & -21 & 0 & 4 \end{bmatrix} /4 \tag{14}$$

$$G = diag(S_G) V_{6\times3} = \begin{bmatrix} 1 & 0 & 0 \\ -2/9 & -2/9 & -2/9 \\ -2/9 & 2/9 & -2/9 \\ 1/90 & 1/45 & 2/45 \\ 1/90 & -1/45 & 2/45 \\ 32/45 & 16/45 & 8/45 \\ 32/45 & -16/45 & 8/45 \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

1.2. Learned Winograd scales

Table 1 shows the difference in values between the standard Winograd scales and our learned Winograd scales for $F(6,3)$. It is worth noting that the magnitudes of S_A have become smaller while those of S_G are bigger, and S_B stays relatively unchanged.

2. Comparison with learning transformation matrices instead of Winograd scales

[5] proposed to treat the Winograd transformation matrices A, B, and G as learnable parameters and jointly optimize them with other model weights and biases in a QAT setup. Although this is much less practical in the domain of Generative AI, as mentioned above, we still adopt this paradigm to compare with our method. All training setups are the same except that transformation matrices A, B, and G are learned directly using random noise inputs instead of learning only the scaling factors S_A, S_B, and S_G. The results are shown in Table 2 using the InstaFlow-0.9B model with AKL autoencoder. It can be seen that learning the transformation matrices directly offers almost no improvement compared to the standard Winograd transforms. This could be due to the use of random noise as layer inputs and treating each layer independently.

Table 1. Comparison between standard Winograd scales and our learned Winograd scales.

tile size	Standard Winograd scales			Learned Winograd scales		
	S_A	S_B	S_G	S_A	S_B	S_G
$F(6,3)$	1	1	1	0.525	-1.378	-1.382
	1	-4.5	-0.222	0.354	-4.908	-0.576
	1	-4.5	-0.222	0.351	-4.912	-0.579
	1	90	0.0111	0.0601	90.366	0.184
	1	90	0.0111	0.0609	90.362	0.182
	1	1.406	0.711	0.491	1.820	1.119
	1	1.406	0.711	0.490	1.823	1.120
	1	1	1	0.519	1.386	1.391

Table 2. Results on the InstaFlow-0.9B model with group-wise quantization, Winograd convolution, and AKL autoencoder. Comparison between learning scales and learning transformation matrices.

IF-0.9B, COCO2017-5k, AKL				
Model	tile size	Bits	FID(↓)	CLIP(↑)
FP16	N/A	16/16	23.00	30.19
W4A8	N/A	4/8	28.73	29.09
W8A8	N/A	8/8	23.04	30.16
W8A8 Winograd	F(4,3)	8/8	217.16	15.14
Standard scales	F(6,3)	8/8	326.96	5.95
W8A8 Winograd	F(4,3)	8/8	24.51	29.87
Learned scales	F(6,3)	8/8	26.58	29.65
W8A8 Winograd	F(4,3)	8/8	204.00	17.13
Learned transforms	F(6,3)	8/8	371.86	3.38

3. Advantage of data-free approach over calibration data and other Winograd methods

Using our paradigm, we fine-tuned Winograd scales and transformation matrices in both end-to-end and BRECQ [8] modes. We compute loss using the difference between the generated images or features of the fully group-wise quantized model and its FP16 counterpart. We randomly select 10k prompts from the poloclub/diffusiondb [13] dataset for calibration and use the same setup as when training with

random noise. It is more challenging to set up an end-to-end training pipeline, and the diffusion model as a whole needs to be kept on the GPU, hence requiring longer training time and more memory. It can be seen in Table 3 that learning scales with calibration data is also effective for $F(4,3)$, but less so for $F(6,3)$ than learning with noise. The poor accuracy for calibration data when compared to our method may be attributed to the small sample size of 10k prompts, but using more samples may result in overfitting. Similar to data-free training, learning Winograd transforms with calibration data failed to produce good-quality generations. Furthermore, training with BRECQ takes significantly longer ($> 10\times$) due to a much more complex training pipeline, with each layer having its own trained Winograd scales, as opposed to a single set of Winograd scales for all layers of one network in our method.

Channel balancing (BQW) [4] only quantizes the Hadamard product, which we could achieve using only group-wise quantization. PAW+FSQ [3] is perhaps not suitable for large-scale diffusion models: transformation matrices are difficult to finetune with the use of calibration data, as shown in Table 3. The use of calibration data for PAW can make it difficult to ensure generalizability to unseen downstream tasks for foundation models. FSQ optimizes hadamard product output (Y) quantization, whereas our learned scales jointly optimize the entire Winograd algorithm, including transformation matrices and Y.

4. Comparison of group-wise quantization against other quantization methods

Table 4 and Table 5 show the comparison between our group-wise quantization method against a popular, recently proposed quantization scheme, called Q-Diffusion[7], for Stable Diffusion V1.4[11] with DPMSolver++[10] and PLMS[9] sampler, respectively. All models were sampled for 25 steps, and MSCOCO 2017 was used to generate FID and CLIP scores. We conducted the experiments with Q-Diffusion using the official codebase and pre-calibrated quantized checkpoints released by the authors.

Figure 1, 2, 3, 4, 5, and 6 compare more qualitative examples generated by the 8-bit Winograd convolution with those generated by the full-precision, 4-bit, and 8-bit models with standard convolution. We also show the images generated by 8-bit Q-diffusion using standard convolution. It can be observed that W8A8 group-wise quantized Stable Diffusion can generate images of nearly identical quality compared with either the FP16 or Q-Diffusion model while being calibration-free. For W4A8, group-wise quantization shows no significant drop in image generation quality and text image alignment.

Table 3. Train Winograd scales and transforms using calibration data.

IF-0.9B, COCO2017-5k, AKL				
Model	tile size	Bits	FID(↓)	CLIP(↑)
Learned scales noise	F(4,3)	8/8	24.51	29.87
	F(6,3)	8/8	26.58	29.65
Learned scales Calibration data E2E	F(4,3)	8/8	24.48	29.85
	F(6,3)	8/8	36.18	29.28
Learned transforms Calibration data E2E	F(4,3)	8/8	138.94	21.05
	F(6,3)	8/8	262.10	14.20
SD-1.4, COCO2017-5k, AKL, DPMSolver++				
FP16	N/A	16/16	21.63	31.72
Learned scales noise	F(6,3)	8/8	20.75	31.57
Learned scales BRECQ	F(6,3)	8/8	170.24	23.24

Table 4. Results on the Stable Diffusion V1.4 model with group-wise quantization, AKL autoencoder and DPMSolver++ sampler with 25 steps. Comparison with the results from Q-Diffusion [7].

SD-1.4, COCO2017-5k, AKL, DPMSolver++				
Model	tile size	Bits	FID(↓)	CLIP(↑)
FP16	N/A	16/16	21.63	31.72
W4A8	N/A	4/8	21.21	30.84
W8A8	N/A	8/8	21.52	31.70
W8A8 Q-Diffusion [7]	N/A	8/8	21.03	31.16

Table 5. Results on the Stable Diffusion V1.4 model with group-wise quantization, AKL autoencoder and PLMS sampler with 25 steps. Comparison with the results from Q-Diffusion [7].

SD-1.4, COCO2017-5k, AKL, PLMS				
Model	tile size	Bits	FID(↓)	CLIP(↑)
FP16	N/A	16/16	22.94	31.74
W4A8	N/A	4/8	22.19	30.01
W8A8	N/A	8/8	22.48	31.68
W8A8 Q-Diffusion [7]	N/A	8/8	24.42	31.07

5. Transferability of learned Winograd scales across datasets

Because Winograd scales are learned from random noise inputs in our method, the same Winograd scale used for one dataset should be transferable to other datasets. Table 6 shows the accuracy results for the CIFAR10 dataset, whereas the same Winograd scale values learned for the ImageNet dataset models previously are applied to the CIFAR10 dataset models directly. Winograd, using learned scales, can successfully restore the accuracy of the full-precision ResNet networks for both tile sizes. This confirms the transferability of our learned Winograd scales across datasets. Table 7 shows the results from Stable Diffusion V1.5 with learned Winograd scales, using DPM-

Table 6. ResNets Top-1 accuracy results on the CIFAR-10 dataset.

Model	tile size	Bits	Standard Winograd	Winograd with Learned Scales
ResNet18	F(4,3)	W8/A8	92.55%	93.00%
93.07%	F(6,3)	W8/A8	33.62%	92.70%
ResNet34	F(4,3)	W8/A8	93.05%	93.26%
93.33%	F(6,3)	W8/A8	22.90%	92.90%
ResNet50	F(4,3)	W8/A8	93.49%	93.70%
93.65%	F(6,3)	W8/A8	81.13%	93.56%

Table 7. Results on the Stable Diffusion V1.5 model with group-wise quantization, AKL autoencoder and DPMSolver++ sampler with 25 steps.

SD-1.5, MJHQ-2k, AKL, DPMSolver++				
Model	tile size	Bits	FID(↓)	CLIP(↑)
FP16	N/A	16/16	41.40	25.78
Learned scales noise	F(4,3)	8/8	44.03	25.53
	F(6,3)	8/8	45.56	25.56

Solver++ sampler with 25 sampling steps and a classifier-free guidance scale of 5.0 on the MJHQ dataset. The same learned Winograd scales work for both the COCO and MJHQ datasets.

6. Comparison to prior QAT works for fully quantizing Winograd

In comparison to previous QAT studies [5] that achieve full quantization by learning Winograd transformation matrices, we can achieve comparable results by fine-tuning only the Winograd scales. As shown in Table 6, 8-bit ResNet18 with Winograd achieves an accuracy of 93% for $F(4,3)$ (an accuracy drop of 0.07% in comparison to the full-precision model) using our learned scales, whereas Winograd-aware QAT [5] observes an accuracy drop of 0.7% for the similar setting.

7. Highly optimized kernels design for text-to-image generation inference on CPUs

While C/C++ runtimes like stablediffusion.cpp [1] demonstrate performance and potential on CPUs, the baseline group quantized kernels have significant compute overheads (the leftmost bar in Figure 6(a)). As a result, a higher proportion of compute instructions do not perform multiplies, i.e., real work, rendering them unsuitable for meeting the required latency requirements for text-to-image generation model variants deployed on commodity CPUs.

To reduce compute overhead and increase the reuse of the input matrices, as well as the use of MAC and vector operations, our highly optimized matrix multiply kernels consider a series of consecutive rows and columns from the two input matrices at once. The use of multiple rows and columns allows for greater reuse of quantized matrices and associated scale factors, as well as fewer load operations. Furthermore, to reduce the high overhead from reduction operations and quantized operand unpacking operations in group-quantized matrix multiply kernels, as well as improve MAC unit utilization and thus the percentage of useful work, our highly optimized kernels perform the following optimizations:

- Amortize the cost of loading operands and the cost of weight unpacking across multiple output channels, resulting in fewer load operations, increased operand reuse, and greater use of vector operations.
- Eliminate the overhead of frequently occurring explicit reduction operations in the matrix multiply kernel (required to accumulate partial dot products from different vector lanes of SIMD fused multiply-accumulate (dot product) operations to obtain the final dot product result for a quantized weight group) of group-wise quantized kernels by ensuring that different vector lanes of dot product instructions operate on weight elements and groups from different output channels so that dot product instructions can take advantage of their implicit accumulate (reduction) operations.
- Solve the resulting non-contiguous access patterns of quantized groups across channels in memory by reformatting weights offline to match the compute order and packing, then storing them in memory in reordered format.
- Maximize the use of the available high MAC throughput matrix-matrix multiply-accumulate instruction, which can perform twice as many MAC operations when compared to an equivalent SIMD dot product instruction.

The benefits of our matrix multiply kernel optimizations for group-wise quantized Winograd convolution and attention layers for Arm CPUs will easily extend to GPUs, neural engines, and x86 processors for the following reasons: (1) Our kernels reduce the number of instructions in the critical path of the group-wise quantized kernels through efficient aligning and reordering of operands of group-quantized formats to take advantage of implicit reduction operations of vector dot product instructions, increase operand reuse, maximize MAC unit utilization, minimize overhead and memory accesses, (2) The vector and matrix-matrix multiply operations used in optimized group-wise quantized kernels are available in other existing CPU and GPU architectures, so there is no need to add new instructions to the CPU/GPU ISA.

Our code is available at https://gitlab.arm.com/artificial-intelligence/efficientwinograd.

References

[1] stable-diffusion.cpp. 4

[2] Syed Asad Alam, Andrew Anderson, Barbara Barabasz, and David Gregg. Winograd convolution for deep neural networks: Efficient point selection, 2022. 1

[3] Tianqi Chen, Weixiang Xu, Weihan Chen, Peisong Wang, and Jian Cheng. Towards efficient and accurate winograd convolution via full quantization. In *Advances in Neural Information Processing Systems*, 2023. 3

[4] Vladimir Chikin and Vladimir Kryzhanovskiy. Channel balancing for accurate quantization of winograd convolutions. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1, 3

[5] Javier Fernandez-Marques, Paul Whatmough, Andrew Mundy, and Matthew Mattina. Searching for winograd-aware quantized networks. *Proceedings of Machine Learning and Systems*, 2020. 2, 4

[6] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[7] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. 3

[8] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. {BRECQ}: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*, 2021. 2

[9] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds, 2022. 3

[10] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models, 2023. 3

[11] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3

[12] Kevin Vincent, Kevin Stephano, Michael A. Frumkin, Boris Ginsburg, and Julien Demouth. On improving the numerical stability of winograd convolutions. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. 1

[13] Zijie J. Wang, Evan Montoya, David Munechika, Haoyang Yang, Benjamin Hoover, and Duen Horng Chau. DiffusionDB: A large-scale prompt gallery dataset for text-to-image generative models. *arXiv:2210.14896 [cs]*, 2022. 2

Figure 1. InstaFlow-0.9B with AKL. Prompt *"A puppy wearing a hat; realistic"*

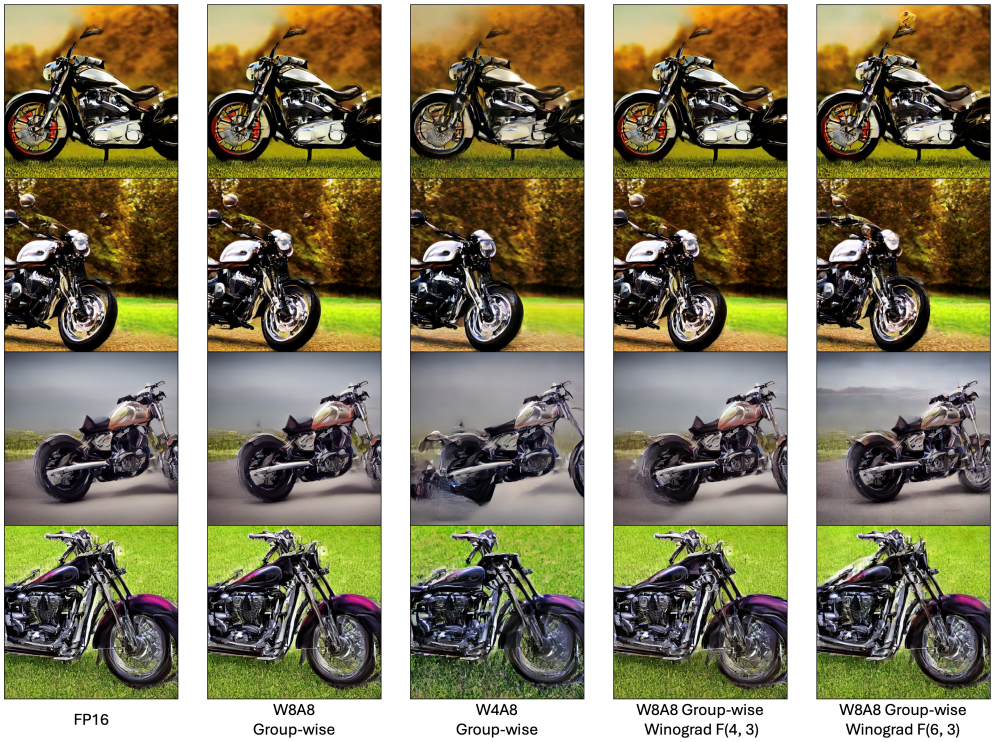

Figure 2. InstaFlow-0.9B with AKL. Prompt *"A shiny motorcycle on the field; realistic, high-resolution"*

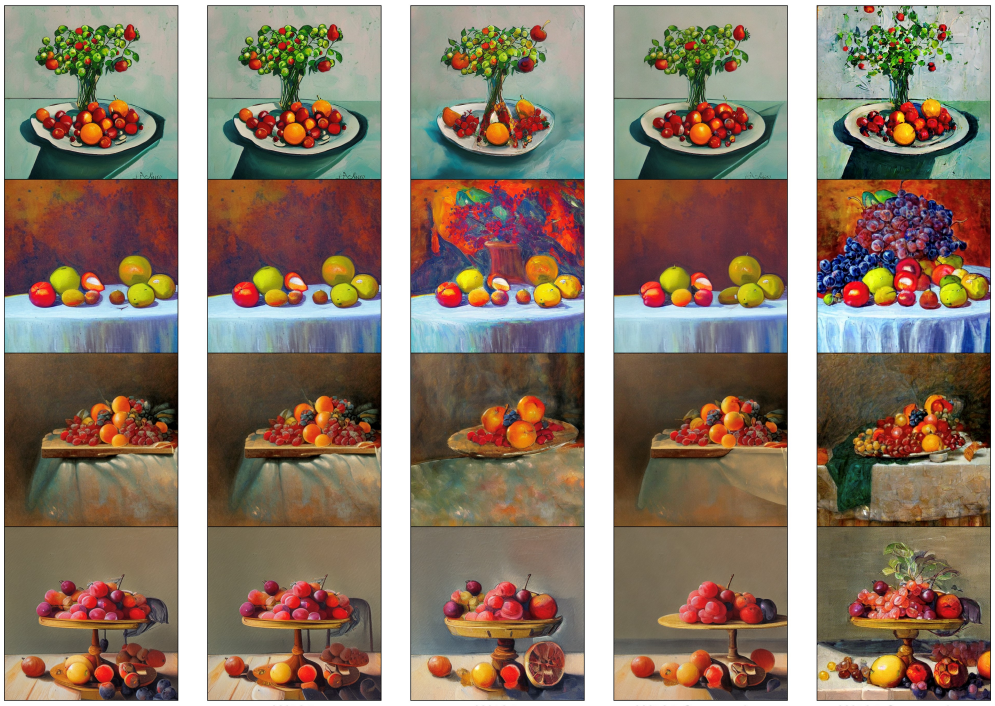

Figure 3. Stable Diffusion V1.5 with AKL and DPMSolver++ sampler. Prompt *"A painting of a table with fruit on top of it"*

Figure 4. Stable Diffusion V1.5 with AKL and DPMSolver++ sampler. Prompt *"A realistic photo of a lovely cat"*

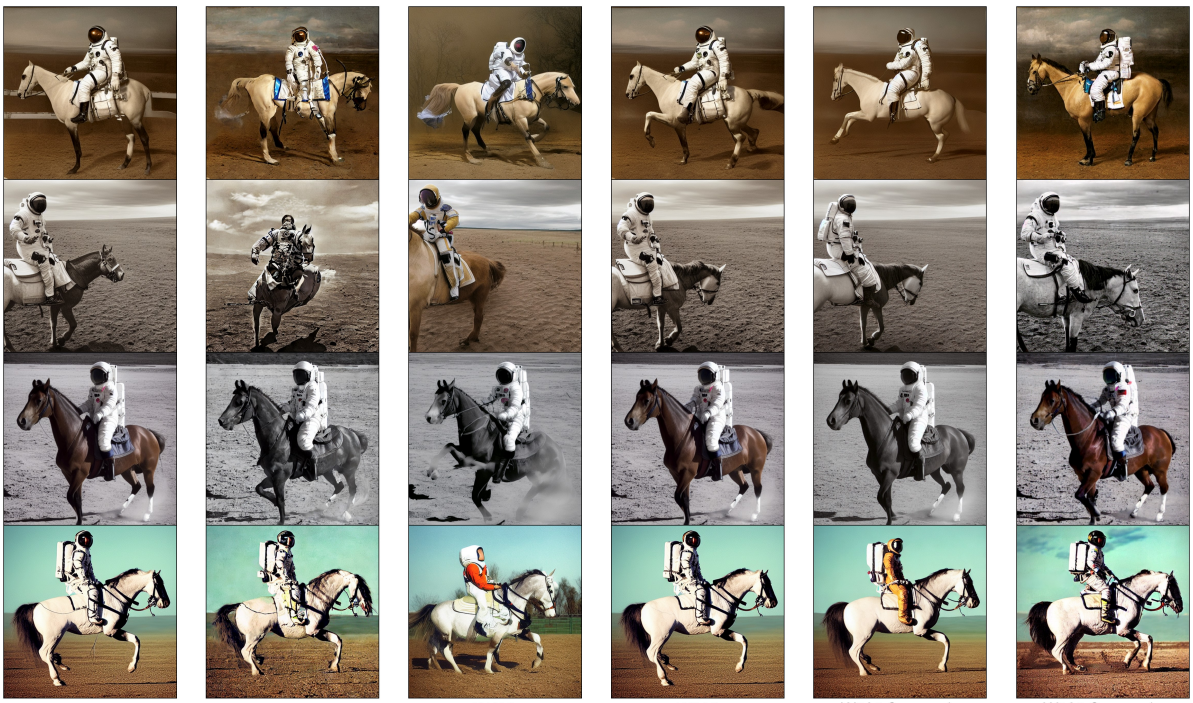

Figure 5. Stable Diffusion V1.4 with AKL and DPMSolver++ sampler. Prompt *"A photograph of an astronaut riding a horse"*

Figure 6. Stable Diffusion V1.4 with AKL and DPMSolver++ sampler. Comparison with Q-Diffusion. Prompt *"A building wall and pair of doors, along with vases of flowers on the outside of the building"*