

A Technical Appendices and Supplementary Material

A.1 RMSNorm Backward Pass

For every vector $\mathbf{x} \in \mathbf{R}^d$, we define

$$f(x) = \frac{x}{\|\mathbf{x}\|} = \frac{x}{\sqrt{\sum_{i=1}^d x_i^2}} = \frac{x}{\sqrt{\mathbf{x}^\top \mathbf{x}}}. \quad (2)$$

Given a matrix \mathbf{X} , the $\text{RMSNorm}(\mathbf{X})$ applies Equation (2) on each row of \mathbf{X} in the forward pass. For a given orthogonal transformation matrix \mathbf{Q} , we will have $\mathbf{f}(\mathbf{Q}\mathbf{x}) = \mathbf{Q}\mathbf{f}(\mathbf{x})$ as $\|\mathbf{Q}\mathbf{x}\| = \sqrt{\mathbf{x}^\top \mathbf{Q}^\top \mathbf{Q} \mathbf{x}} = \|\mathbf{x}\|$, which is known as the *distributive property*. Using this fact, Ashkboos et al. [2] showed that any orthogonal transformation \mathbf{Q} can be fused to the weights of the previous layer, and RMSNorm will preserve it because of the distributive property above. This approach eliminates the overhead of applying Hadamard transformations during the forward pass.

Now, consider the derivative of $\mathbf{f}(\mathbf{x})$ which is used during the backward pass of fine-tuning. For a given input vector $\mathbf{x} \in \mathbf{R}^d$, we will have

$$\frac{\partial}{\partial x} \left(\frac{x}{\|\mathbf{x}\|} \right) = \frac{1}{\|\mathbf{x}\|} \left(I - \frac{\mathbf{x}\mathbf{x}^\top}{\|\mathbf{x}\|^2} \right), \quad (3)$$

which **does not** have the above distributive property. This shows that one cannot use the fusion idea [3; 2; 23] to merge the Hadamard transformations into the previous weight matrix during the backward pass, especially as we need to apply left-hand-side Hadamard transformations on the errors (see Figure 1).

A.2 HALO Levels for FFT

We summarize all the HALO levels for fine-tuning of a linear layer in Table 3. We use tensor-wise symmetric quantization function \mathbf{Q} for all data types (input, weights, and errors).

Method	Forward Calculation (F)	Input Gradient Calculation (E)	Weight Gradient Calculation (G)	Notes
No-HALO	$\mathbf{X}_Q \mathbf{W}_Q^\top$	$(\mathbf{E}_Y)_Q \mathbf{W}_Q$	$(\mathbf{E}_Y^\top)_Q \mathbf{X}_Q$	1. Suitable for wide ranges (e.g., FP8).
HALO-1($\mathbf{F}^H, \mathbf{E}^H, \mathbf{G}^H$)	$(\mathbf{X}\mathbf{H})_Q (\mathbf{W}\mathbf{H})_Q^\top$	$(\mathbf{E}_Y)_Q (\mathbf{W}\mathbf{H})_Q \mathbf{H}^\top$	$(\mathbf{E}_Y^\top)_Q (\mathbf{X}\mathbf{H})_Q \mathbf{H}^\top$	1. Outlier mitigation during the forward pass. 2. Easy integration with FSDP with AC. 3. Quantized activation for memory reduction. 4. Suitable for moderate ranges (e.g., FP6).
HALO-2($\mathbf{F}^H, \mathbf{E}^H, \mathbf{G}^H$)	$(\mathbf{X}\mathbf{H})_Q (\mathbf{W}\mathbf{H})_Q^\top$	$\mathbf{H}^\top (\mathbf{H}\mathbf{E}_Y)_Q (\mathbf{W}\mathbf{H})_Q \mathbf{H}^\top$	$(\mathbf{E}_Y^\top)_Q (\mathbf{X}\mathbf{H})_Q \mathbf{H}^\top$	1. Most accurate scheme. 2. Suitable for narrow ranges (e.g., INT8).

Table 3: HALO levels for full fine-tuning (FFT). We use the quantization function \mathbf{Q} for quantizing different data types and perform the computation with low precision. AC stands for Activation Checkpointing.

A.3 LoRA-style Linear Module and HALO for PEFT

A LoRA linear module includes the following operations:

$$\mathbf{Y} = \mathbf{X} \cdot \mathbf{W}^\top + (\mathbf{X} \cdot \mathbf{U}^\top) \cdot \mathbf{V}^\top, \quad (4)$$

$$\mathbf{E}_X = \mathbf{E}_X^{\mathbf{U}\mathbf{V}} + \mathbf{E}_X^{\mathbf{W}} = (\mathbf{E}_Y \cdot \mathbf{V}) \cdot \mathbf{U} + \mathbf{E}_Y \cdot \mathbf{W}, \quad (5)$$

$$\mathbf{G}_V = \mathbf{E}_Y^\top \cdot (\mathbf{X} \cdot \mathbf{U}^\top), \quad (6)$$

$$\mathbf{G}_U = (\mathbf{V}^\top \cdot \mathbf{E}_Y^\top) \cdot \mathbf{X}, \quad (7)$$

where \mathbf{G}_U and \mathbf{G}_V are the gradients of \mathbf{U} and \mathbf{V} , respectively. Since low-rank operations are fast, our goal is to quantize operations not involving the the low-rank matrices \mathbf{U} and \mathbf{V} .

As mentioned in Section 3, we integrate HALO with PEFT by quantizing the majority of the computation while retaining the low-rank computations in high precision. In this case, the Equations (4-5) will become

$$\mathbf{Y} \approx (\mathbf{X}\mathbf{H})_{\mathbf{Q}} \cdot (\mathbf{W}\mathbf{H})_{\mathbf{Q}}^{\mathbf{T}} + (\mathbf{X}\mathbf{U}^{\mathbf{T}}) \cdot \mathbf{V}^{\mathbf{T}}, \quad (8)$$

$$\mathbf{E}_{\mathbf{X}} \approx \mathbf{E}_{\mathbf{X}}^{\mathbf{UV}} + \mathbf{H} \cdot (\mathbf{H}^{\mathbf{T}}\mathbf{E}_{\mathbf{Y}})_{\mathbf{Q}} \cdot (\mathbf{W}\mathbf{H})_{\mathbf{Q}}\mathbf{H}^{\mathbf{T}}. \quad (9)$$

We show the above scheme by **HALO_{PEFT}**.

Next, we compare HALO_{PEFT} against LoRA [19], on GSM8K, SQL, and ViGGO for FP8, FP6 and INT8 in Table 4. For INT8 and FP8, HALO_{PEFT} is always within the standard deviation of baseline. On GSM8K, FP6 has approximately a 2% accuracy degradation, showing the challenge of recovering high-precision accuracy with only 6 bits. For ViGGO and SQL, both INT8 and FP6 recover accuracy with at most a 0.5% average accuracy drop.

Table 4: Single-epoch accuracy comparison between LoRA [19] and HALO_{PEFT}(ours). We use rank $r = 16$ for adapters and apply FP6 (E3M2) and INT8 for quantization.

Precision	Method	GSM8k	ViGGO	SQL
BF16	LoRA	69.4 ± 0.8	94.1 ± 0.2	80.0 ± 0.4
INT8	HALO _{PEFT}	69.0 ± 0.5	93.4 ± 0.7	79.9 ± 0.4
FP6		67.3 ± 0.6	93.6 ± 0.7	79.9 ± 0.5
FP8		69.4 ± 1.0	94.2 ± 0.6	80.0 ± 0.3

A.4 Hadamard Effect

Figure 5 illustrates the effect of applying Hadamard transformations to the right and left sides of the activations (input matrices), errors (gradients with respect to the output), and weights. For the activations, applying Hadamard on the right side effectively removes outliers, whereas for the errors, a left-side Hadamard transformation is necessary to eliminate outliers, as also noted in Figure 1.

A.5 HQ-FSDP Details

Here we discuss some details about our HQ-FSDP implementation. **Master Weights.** In HALO, weights are maintained in BF16. Accordingly, HQ-FSDP stores the parameters in BF16 and applies quantization before communication. **Forward vs. Backward.** Since the weights remain unchanged between the forward and backward passes, the quantized shard could be stored during the forward pass in each process and only communicated during the backward pass. This approach eliminates the need for a second quantization and potential Hadamard transform during the backward pass. However, it introduces additional memory overhead, as the quantized weights must be stored alongside the master BF16 weights. Additionally, since the quantization and Hadamard transform on the weights are distributed across FSDP processes, their overhead is relatively small. Instead, we adopt an intermediate approach: save only the global quantization scales during the forward pass and recompute the quantization and Hadamard transform during the backward pass. **FSDP Wrapping Policy.** Following standard practice, we wrap each transformer block with an FSDP module (FSDP communications happen before each transformer block). However, for HQ-FSDP, we skip the layer-norm modules (keeping full weights in every process) as we do not intend to quantize them. Our experiments show that this only marginally increases memory usage and does not change runtime. **Distributed Hadamard Transformation.** When the scheme requires a right Hadamard transformation, HQ-FSDP applies it in a distributed way; process i performs $\mathbf{W}_i\mathbf{H}$ where \mathbf{W}_i denotes the i 'th shard. However, this requires the weight matrix to be sharded by row, i.e., each row should entirely reside within a single shard. As FSDP requires equally sized shards to be fast, we dynamically insert small dummy parameter tensors of carefully chosen sizes when needed. This guarantees row-aligned sharding without compromising performance.

A.6 Hyper-Parameters

In all experiments, we tune the hyper-parameters on the base BF16 tasks, and re-use the same values for low-precision training. We always perform single-epoch experiments using the AdamW optimizer

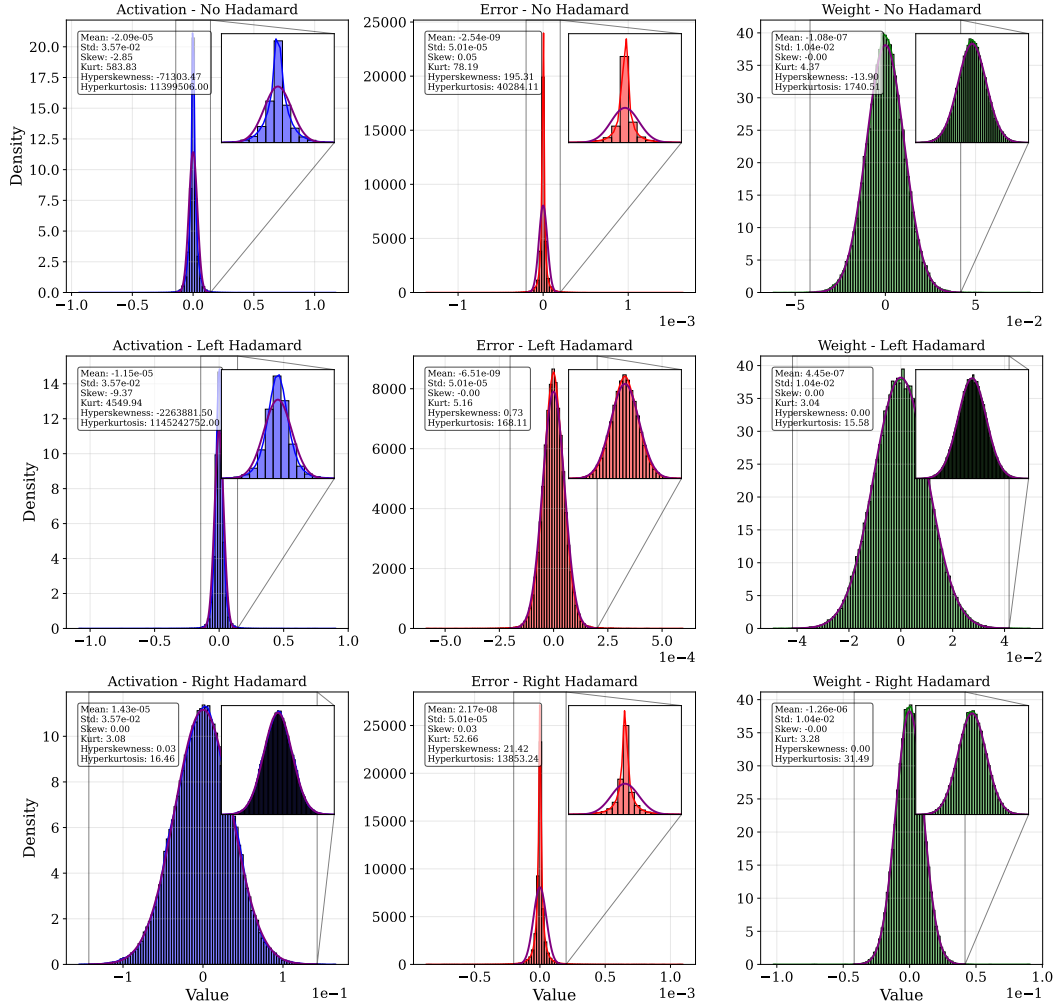


Figure 5: The effect of applying Hadamard transformations on the left and right hand sides of the the activations, errors, and weights.

with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and a linear learning rate warm-up of 20 steps. The batch size and sequence length are fixed at 32 and 512. For FFT, we choose learning rates 4×10^{-5} , 6×10^{-6} , and 3×10^{-5} for ViGGO, GSM8k, and SQL, respectively, and for PEFT LoRA experiments, we choose the learning rate 6×10^{-4} and LoRA rank of 16 for all datasets. These learning rates were found to be the best using a grid search within the range $[10^{-6}, 10^{-3}]$ of 20 uniform log-linearly separated grid points, trained and evaluated using the non-quantized BF16 training precision.

A.7 Ablation study on Hadamard schemes

Figure 6 presents various combinations of applying Hadamard transformations during the forward and backward passes. As shown in Figure 6-Left, applying Hadamard in the forward pass is crucial. For the backward pass, Figure 6-Right indicates that HALO-1($\mathbf{F}^H, \mathbf{E}^H, \mathbf{G}^H$) and HALO-2($\mathbf{F}^H, \mathbf{E}^H, \mathbf{G}^H$) are the optimal configurations, balancing the minimization of Hadamard transformations with system-level considerations such as memory usage and communication compression.

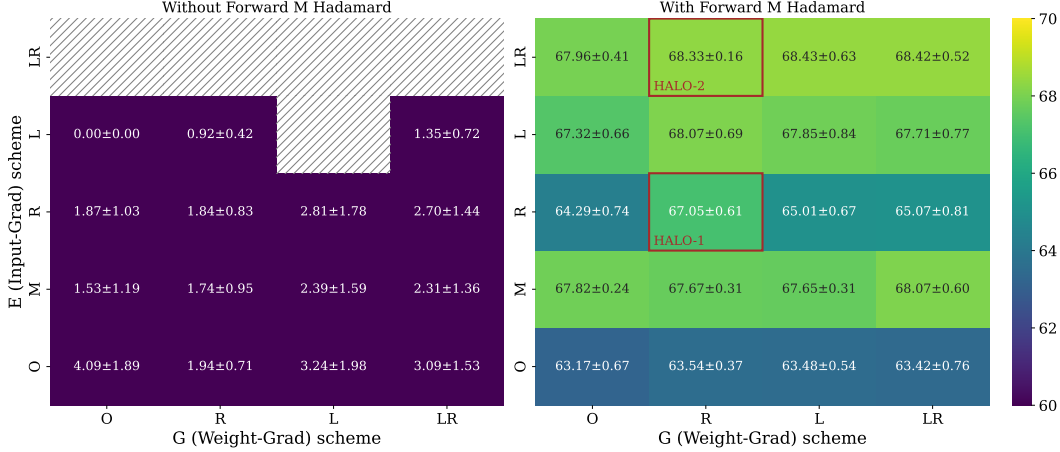


Figure 6: Different combinations of applying Hadamard transformations during the forward and backward passes of training LLAMA3-8B on GSM8K using INT8 precision. Here, "L" denotes applying a Hadamard transformation on the left-hand side, "R" indicates applying it on the right-hand side, and "M" refers to the middle case. "LR" represents applying Hadamard transformations on both the left and right sides, while "O" denotes the case where no Hadamard transformation is applied.

A.8 Ablation Study on HALO Levels

One interesting question concerns the comparison between the different levels of HALO introduced in Section 3.3, that is HALO-0(F, E, G), HALO-1(F^H, E^H, G^H), and HALO-2(F^H, E^H, G^H) used during the fine-tuning of FP8, FP6, and INT8, respectively.

Table 5 shows the effect of using different levels for INT8 and FP6, illustrating the natural finding that a higher HALO level results in higher final test accuracy. For INT8 precision, HALO-0 fails to recover accuracy, leading to an approximate 40% drop in accuracy (on average) on our datasets. At the next level, HALO-1 recovers approximately 37% of the above accuracy gap by applying right-hand-side Hadamard transformations on the weights and inputs during both the forward and backward passes. Finally, HALO-2 applies left-hand-side Hadamard transformations on the errors, achieving within 1% of the BF16 accuracy. For FP6 precision, although HALO-2 achieves higher accuracy on the GSM8K dataset, believe HALO-1 provides a better trade-off between accuracy and the number of Hadamard transformations.

Table 5: The accuracy effects of using different HALO levels within each quantization precision. The selected level for each precision is presented with **bold** text. We exclude FP8 experiments as HALO-0 recovers the BF16 accuracy.

Precision	Method	GSM8k	ViGG0	SQL
BF16	Baseline	69.26 ± 0.51	94.02 ± 0.29	79.83 ± 0.49
FP6	HALO-0	62.32 ± 0.65	92.92 ± 0.73	79.24 ± 0.36
	HALO-1	66.54 ± 0.22	93.56 ± 0.38	80.20 ± 0.26
	HALO-2	67.42 ± 0.99	93.56 ± 0.38	80.00 ± 0.62
INT8	HALO-0	4.50 ± 1.03	55.98 ± 20.89	74.73 ± 0.45
	HALO-1	62.27 ± 0.64	93.23 ± 0.45	79.43 ± 0.59
	HALO-2	68.15 ± 0.08	93.79 ± 0.08	80.12 ± 0.31

A.9 HALO Inference Speedups

We present HALO as a low-precision fine-tuning method. However, it can be presented as a QAT scheme where the inference of the fine-tuned model will be done in low precision. To this end, following QuaRot [3], we fuse the Hadamard transformations into the previous linear modules and just apply two Hadamards before out-projection and down-projection layers in the Attention and MLP modules. Table 6 shows the inference speedups of a single Transformer block when we

use HALO for INT8 fine-tuning. As expected, the speedups increase with batch size, peaking at a batch size of 8. However, with a batch size of 16, the multi-head attention module becomes another bottleneck, leading to reduced speedup gains.

Table 6: Inference runtimes of one Transformer block in LLAMA3-8B Model, fine-tuned with HALO-2($\mathbf{F}^H, \mathbf{E}^H, \mathbf{G}^H$) with different batch sizes (BS). We use 512 sequence length.

	BS	BF16	HALO	Speedup
	2	3.21ms	2.20ms	1.46×
	4	6.19ms	3.28ms	1.89×
	8	13.12ms	6.88ms	1.91×
	16	26.52ms	14.32ms	1.85×

A.10 Transformer Block Speedups

	INT8				FP8			
	BS=4	BS=8	BS=16	BS=32	BS=4	BS=8	BS=16	BS=32
Ideal	1.62×	1.69×	1.75×	1.70×	1.32×	1.37×	1.35×	1.28×
HALO-0	1.52×	1.63×	1.67×	1.63×	1.27×	1.32×	1.32×	1.24×
HALO-1	1.26×	1.43×	1.51×	1.50×	1.07×	1.19×	1.22×	1.16×
HALO-2	1.15×	1.31×	1.36×	1.38×	0.99×	1.11×	1.12×	1.10×

Table 7: HALO speedups for different batch sizes (BS) on three consecutive **decoder blocks** of LLAMA3-8B model with 512 sequence length on a single RTX 4090. Ideal shows the speedups when there is no quantization and Hadamard overheads. We **bold** the chosen scheme for each precision.

We evaluate using HALO on three consecutive LLAMA3-8B Transformer blocks (the largest number of blocks that fit on one GPU with batch size 32). Table 7 shows the speedup numbers for different levels in HALO when we apply INT8 and FP8 precisions. Using INT8, the most accurate HALO level, HALO-2, achieves speedups of 1.15× to 1.38× compared to BF16 when using our kernels. The speedup increases with larger batch sizes, as the quantization and Hadamard overheads become less significant, and the matrix multiplications become the primary bottleneck. For FP8, we use HALO-0, which achieves speedups of 1.27× to 1.32×, coming within 5% of the ideal speedup. We note that since FP6 has the same TensorCore peak performance as FP8 (with less read/write overhead), the speedups achieved with FP8 can be considered a lower bound for the potential speedups with FP6 as well.

A.11 FP8 Linear Layer Speedup

We also benchmark HALO-0 and HALO-1 with FP8 quantization in Table 8. HALO-0 is nearly on par with ideal speedup, peaking at 1.68×. We also provide HALO-1 results when we use with FP6 precision. However, since hardware support for FP6 matrix multiplication is unavailable, we use FP8 matmul instead to provide a lower bound on the FP6 speedup, which peaks at 1.52×.

Table 8: Forward + backward speedups (over BF16) of a linear layer (4096×4096) across batch sizes (BS) when we quantize inputs, weights, and output gradients using FP8 representation.

(RTX-4090)	BS=4	BS=8	BS=16	BS=32
Ideal	1.20×	1.65×	1.70×	1.78×
HALO-1	1.07×	1.47×	1.48×	1.52×
HALO-0	1.12×	1.62×	1.63×	1.68×

A.12 HQ-FSDP Speedups

In Table 9 we include detailed speedup numbers for FP8 HALO-0, FP8 HALO-1 and INT8 HALO-2 on four RTX 4090 GPUs, with and without HQ-FSDP.

	w/o HQ-FSDP				w/ HQ-FSDP			
	BS=4	BS=8	BS=16	BS=32	BS=4	BS=8	BS=16	BS=32
FP8 Ideal	1.00×	1.01×	1.06×	1.14×	1.39×	1.37×	1.31×	1.28×
FP8 HALO-0	1.00×	1.01×	1.05×	1.12×	1.39×	1.37×	1.30×	1.26×
FP8 HALO-1	0.98×	0.99×	1.02×	1.08×	1.37×	1.34×	1.25×	1.21×
INT8 Ideal	1.00×	1.02×	1.14×	1.43×	1.40×	1.40×	1.48×	1.65×
INT8 HALO-2	0.99×	1.00×	1.09×	1.29×	1.37×	1.34×	1.36×	1.45×

Table 9: HALO speedups for different batch sizes (BS) on three consecutive decoder blocks of LLAMA3-8B model with 512 sequence length on four RTX 4090 GPUs, with and without HQ-FSDP. Ideal shows the speedups when there are no quantization and Hadamard overheads.

A.13 Qwen Results

To evaluate the effectiveness of HALO on larger models, we extend our GSM8k INT8 experiments to the Qwen-2.5 14B and Qwen-2.5 32B models [44]. Table 10 compares HALO-2 against JetFire [41] and the BF16 baseline. This table shows that HALO-2 outperforms both JetFire and BF16 across both model sizes.

Table 10: Comparison of HALO-2 with JetFire and BF16 on GSM8k for Qwen models.

Method	Qwen-14B	Qwen-32B
BF16	81.046 \pm 0.5	86.808 \pm 0.3
JetFire	81.92 \pm 1.12	87.1 \pm 0.71
HALO-2	82.33 \pm 0.68	88.40 \pm 0.8

A.14 Pre-training Results

We apply HALO to pre-training of TinyLlama-1.1B [47] on the C4 dataset [18]. We select a random Chinchilla-optimal subset [18] (22B tokens), and follow the same hyper-parameters as the original TinyLlama-1.1B [47]. Applying combinations of HALO levels and precisions (INT8, FP4, FP6, and FP8), our results can be summarized as follows:

- FP8 closely matches the base BF16 training even with HALO level 0, both achieving a evaluation loss of 2.55.
- FP6 converges only with HALO level 2, achieving a final evaluation cross-entropy of 2.70.
- INT8 and FP4 diverge, no matter the HALO level.

A.15 MXFP6 Results

In this section, we apply HALO-1 and HALO-2 on the MXFP6 format, comparing them with pure MXFP6 and BF16 training. We consider the LLAMA3-8Bmodel [13] and all three datasets mentioned in the main text. Table 11 shows that HALO-2 closes the accuracy gap with BF16 on the GSM8k [8] dataset.

Table 11: MXFP6 fine-tuning results for LLAMA3-8B[13] on the three datasets, with and without HALO.

Method	GSM8k	ViGGO	SQL
BF16	69.3 \pm 0.5	94.0 \pm 0.3	79.9 \pm 0.5
MXFP6	67.8 \pm 0.2	93.6 \pm 0.6	79.6 \pm 1.0
MXFP6-HALO1	68.0 \pm 0.3	93.7 \pm 0.3	80.5 \pm 0.2
MXFP6-HALO2	68.9 \pm 0.5	93.5 \pm 0.5	80.3 \pm 0.4