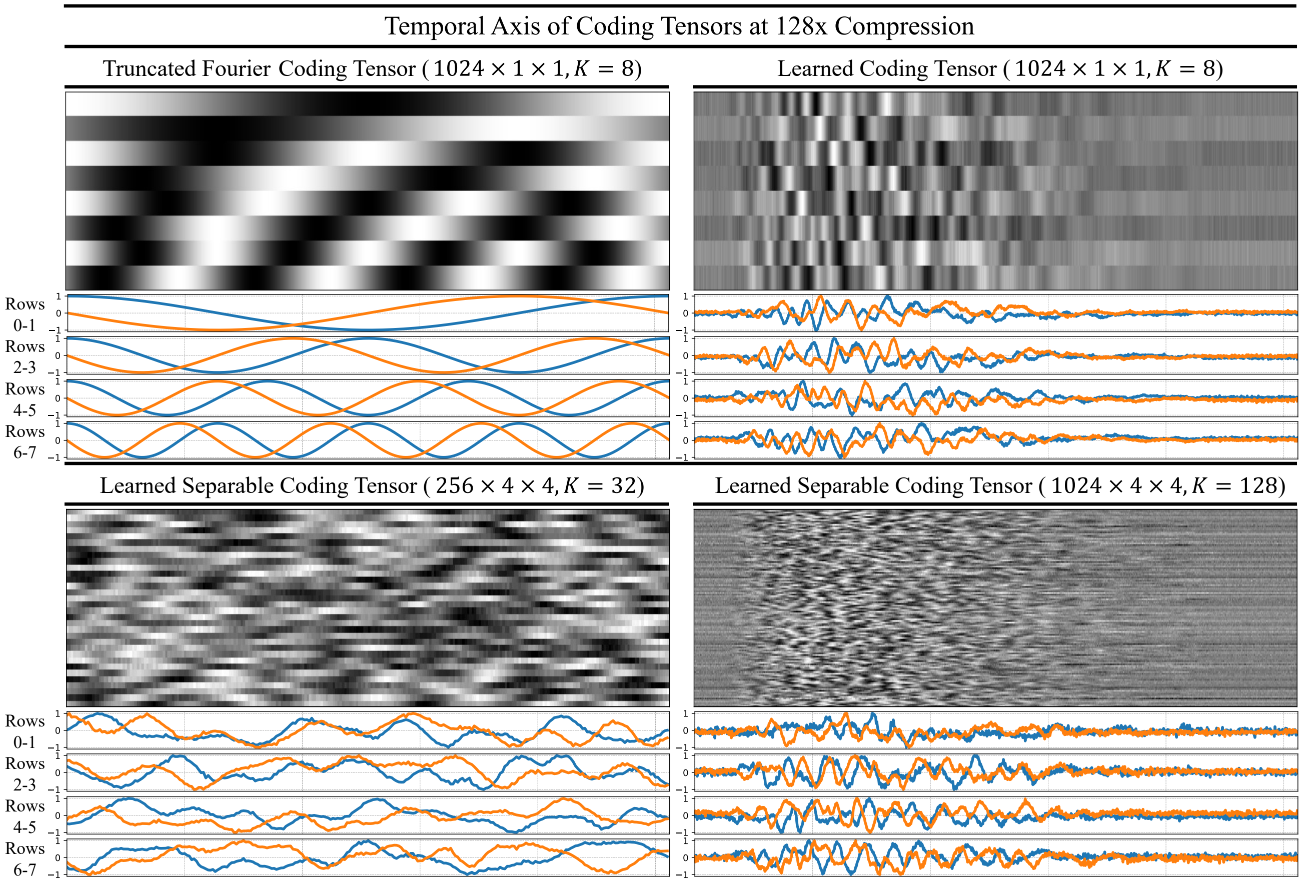
Supplementary Document for “Learned Compressive Representations for Single-Photon 3D Imaging”

S. 1. Dataset Bias in Learned Coding Tensors with $M_t = N_t$

In this section, we analyze the training dataset depth bias that is embedded in some coding tensor designs, and its effect on generalization to scenes with depths that appear less often in the dataset.

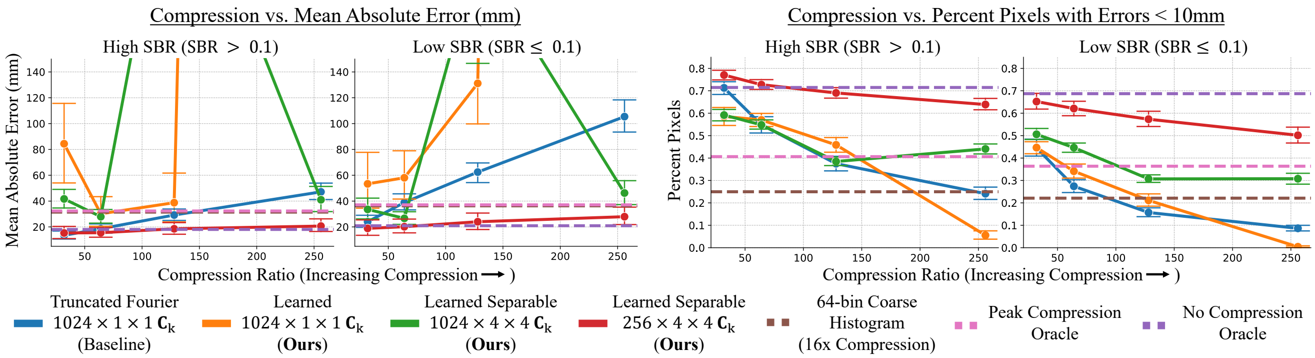
Depth Range Bias in Learned Coding Tensors: Supplementary Figures 1 and 17 visualize the temporal dimension for different coding tensor designs as a matrix. The learned coding tensors that operate on the full temporal dimension (i.e., $M_t = N_t = 1024$) show structure in approximately the first half of the matrix, and in the second half their magnitude is close to 0. According to the coding tensor design heuristics introduced in [6], these coding tensors may have depth ambiguities in the second half of the depth range and may not be robust to noise when estimating depths in that range. These learned coding tensors are consistent with the depth range observed in the NYUv2 training dataset whose depths are concentrated between 0.5-7m (i.e., close to half of the depth range in our simulation) [9]. Our evaluation in the main paper only included test scenes with depths < 7m, therefore, this bias had little impact on the performance. To further analyze the impact of this bias on generalization, in the remainder of this section we evaluate the models on a modified Middlebury test set that contains a global depth offset of 7m, making its depth range 7-10m.



Supplementary Figure 1. **Temporal Dimension of C at 128x Compression** Visualization of the temporal dimension for different coding tensors that achieve 128x compression. The matrix visualized for coding tensors of dimension $1024 \times 1 \times 1$ (columns 1 and 2) is an 8×1024 matrix, since $K = 8$ and $M_t = 1024$. On the other hand, the matrix for the learned separable $256 \times 4 \times 4$ $\mathbf{C}_k$ is 32×256 since $K = 32$ and $M_t = 256$. Similarly, the matrix for the learned separable $1024 \times 4 \times 4$ $\mathbf{C}_k$ is a 128×1024 matrix. The bias on the learned coding tensors with $M_t = 1024$ is displayed on the weights whose magnitude is close to 0 on the right-most side of the matrices. The functions shown below each matrix correspond to different rows of the matrix.

Quantitative Performance Analysis on Depths Between 7-10m: Fig. 2 shows how the performance of different compressive histogram models varies as a function of the compression ratio. The learned coding tensors with $M_t = N_t = 1024$

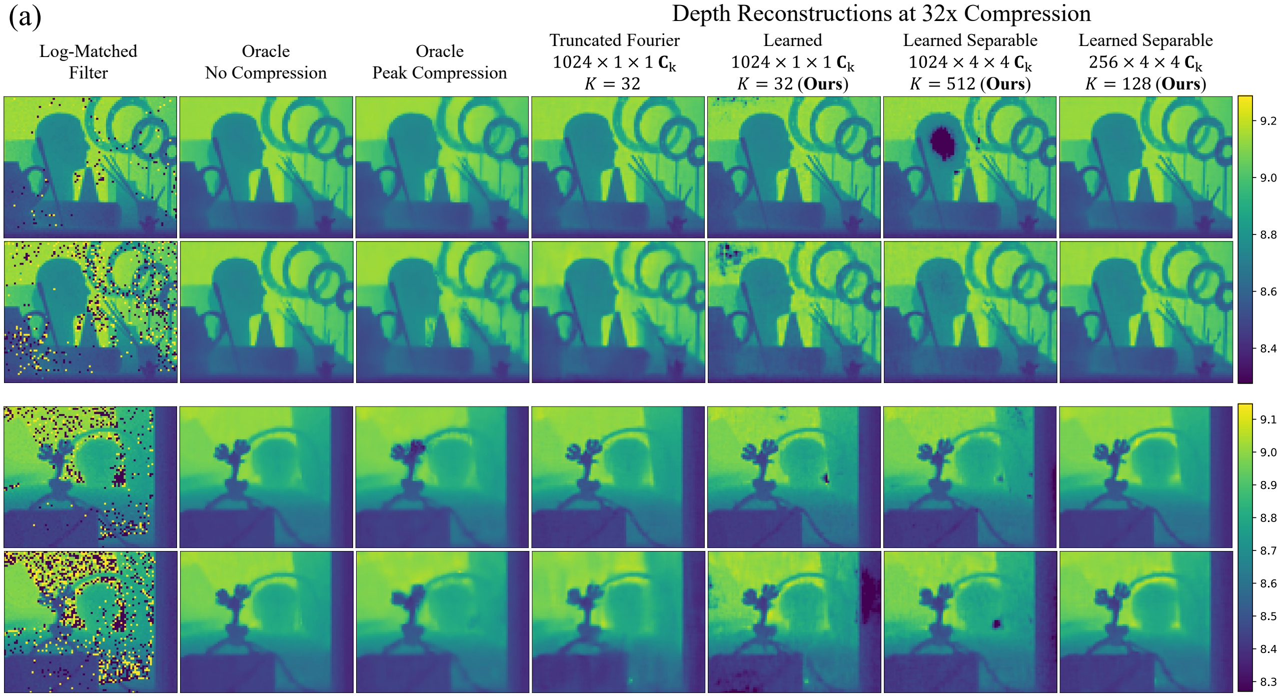
(i.e, orange and green lines) consistently display poor performance across all compression levels. Moreover, the variance in their MAE is very high which is likely due to generalization artifacts. Fourier-based coding tensors (blue line) continue to achieve reasonable performance at CR < 64 and poor performance for higher compression levels, which is consistent with the results in the main paper. Finally, the learned coding tensor with $M_t < N_t$ (red line) displays good performance across all compression and SBR levels, comparable to the results in the main paper, despite the aforementioned dataset bias.

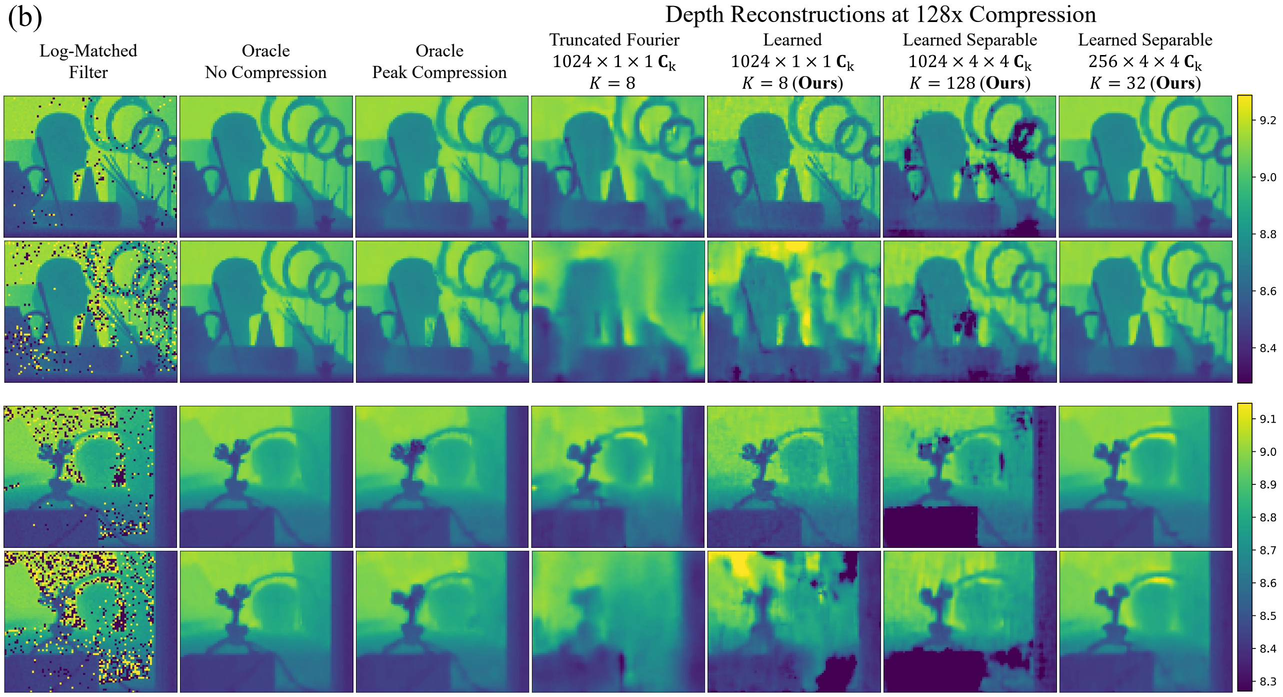


Supplementary Figure 2. **Compression vs. Test Set Metrics on Large Depths Test Set.** Performance on the Middlebury test set with a 7 meter depth offset applied to all depth images before simulation. The two left-most plots show the mean absolute error computed over the test set as we increase compression. Similarly, the two right-most plots show the mean percent of pixels whose absolute depth errors were < 10mm. The simulated test set images were divided into low (SBR ≤ 0.1) and high (SBR > 0.1) SBR groups to be able to disentangle the impact of SBR on the performance of each model. The dashed lines show the peak and no compression baselines whose compression levels do not vary. Each line corresponds to a fixed coding tensor design for which we vary K to control the compression level. Moreover, each point for a given compression level corresponds to a single set of coding tensors jointly optimized with the depth estimation 3D CNN. The learned coding tensors with $M_t = N_t = 1024$ (orange and green lines) show significantly elevated MAE compared to a learned $\mathbf{C}$ with $M_t = 256$. This poor performance is due to the depth reconstruction artifacts observed in Suppl. Fig. 3.

Qualitative Performance Analysis on Depths Between 7-10m: Fig. 3 shows the depth reconstruction for multiple baselines and compressive histograms at 32x and 128x compression. The recovered depth images with learned coding tensors with $M_t = N_t = 1024$ display multiple artifacts at both high (1) and low (0.05) SBR for both scenes. These artifacts explain why we observe elevated MAE in Fig. 2, but at the same time, the percent pixels with errors $< 10mm$ is not incredibly low for some compression levels. On the other hand, the learned separable $256 \times 4 \times 4$ $\mathbf{C}$ not only obtains artifact-free reconstructions but also continues to show the same trends observed in the paper. For instance, at 128x compression and SBR=0.05, it is able to preserve important scene information that is lost when using a Truncated Fourier $\mathbf{C}$.

Summary: In general, it is important to analyze and account for dataset bias in any learning-based model. We find that this bias can lead to learned coding tensors that only work well for a subset of depths. One way to resolve this problem is by augmenting the dataset to include examples with depths for the full depth range. However, we find that an even simpler approach is to consider a coding tensor design that considers a smaller block size, making the tensor convolutional. In this section, we showed that the learned coding tensors that are robust to this dataset bias, continue to provide the same performance benefits that were observed in the main paper.





Supplementary Figure 3. **Depth Reconstructions at 32x and 128x Compression for Scenes with Depths Between 7m and 9.5m.** Recovered depths for two scenes whose depths range between 7m and 9.5m. The compressive histogram models achieve 32x (a) and 128x (b) compression. The SBR levels of 1 and 0.05 correspond to a scene simulated with an average number of detected photons per pixel of [10,10] and [10,200]. Learned coding tensors with $M_t = N_t = 1024$ produce reconstructions with multiple artifacts which indicates that they have poor generalization for this range of depths, which is consistent with the observed depth range bias observed in Suppl. Fig. 1.

S. 2. On-chip Implementation and Power Consumption Analysis

In this section, we provide further details on the UltraPhase SPAD chip used for the power analysis, our compressive histograms implementation on it, and a discussion on the role compressive spatio-temporal histograms can play in the scaling of this in-pixel processing architecture to megapixel SPAD-based 3D cameras with picosecond time resolutions.

UltraPhase SPAD Processing Chip [1]: The UltraPhase chip consists of a 3×6 array of processing cores, each connected to 4×4 SPAD pixels resulting in a 12×24 SPAD camera. Every core is independent, and can execute programs of up to 256 instructions in length at a rate of 140 MOPS and has 4096 bits of available RAM. The system supports a wide range of instructions, from logic to 32-bit arithmetic operations, data manipulation, and custom inter-core synchronization and communication. Although the 3×6 array of processing cores has been fabricated, it has not been 3D stacked on the 12×24 SPAD camera *yet*. Therefore, our analysis focuses on the implementation of our proposed methods on the processing cores and their corresponding power consumption, data rates, frame rates, and processing times.

Implementation: In our current implementation, we assume that one UltraPhase processing core is equivalent to a single SPAD pixel, even though each processing core will eventually be connected to 4×4 SPAD pixels. This assumption is made because the current version of UltraPhase does not keep track of the spatial information of a photon timestamp that was detected within the 4×4 region. This assumption will not be necessary for future versions of UltraPhase or UltraPhase-like chips that are able to keep track of the spatial location of a timestamp that was detected within the SPAD region the processing core is connected to (e.g., [16]). Nonetheless, since there are 3×6 cores, we are able to validate the proposed spatio-temporal compressive histograms since we do know the spatial information of a timestamp at the processing core level.

All methods implemented on UltraPhase take as input 10-bit timestamps. A 10-bit timestamp is consistent with the datasets in the main paper where the full-resolution histogram had $N_t = 1024$ bins, i.e., the detected timestamps could take one of 1024 possible values. We implement the following compressive histogram methods on UltraPhase:

1. **K-bin Coarse Histogram:** Coarse histograms are one of the simplest compressive histograms [6]. When a timestamp is detected, its value is used to determine the histogram bin index to increment. Each coarse histogram bin has a bit depth of 8 bits.

2. **Memory-efficient Truncated Fourier ($1024 \times 1 \times 1$ $\mathbf{C}_k$):** A naive implementation of a Truncated Fourier coding tensor would require storing K $1024 \times 1 \times 1$ coding tensors which would not fit in the memory of one UltraPhase core. However, it is possible to compute the values of any Fourier coding tensor from the first quadrant $[0, \frac{\pi}{2}]$ of the cosine and sine functions using trigonometric identities. The first quadrant corresponds to the one-fourth (first 256 elements) of the first row of the Truncated Fourier matrix shown in Fig. 1. This means that all truncated Fourier coding tensors can be generated from only 256 parameters which occupy 2,048 bits of memory when quantized to 8 bits. When a timestamp arrives, its value is used to generate the K Truncated Fourier coefficients, which are aggregated on a 16-bit compressive histogram. Please refer to [1] for further details on this implementation.

3. **Separable Temporal Fourier and Learned Spatial Coding Tensors ($256 \times 2 \times 2$ $\mathbf{C}_k$):** This compressive histogram model uses the proposed separable coding tensors with dimensions $256 \times 2 \times 2$. The spatial coding tensors ($\mathbf{C}_k^{\text{spatial}}$) are learned, while the temporal coding tensors ($\mathbf{C}_k^{\text{temporal}}$) are fixed to truncated Fourier coding tensors with $M_t = 256$. We leverage the same memory-efficient truncated Fourier implementation described above. All coding tensors are quantized to 8 bits. The resulting spatio-temporal coding tensors occupy 768 bits and 1,024 bits of memory for $K = 8$ and $K = 16$, respectively ($(64 + 2 \cdot 2 \cdot K) \cdot 8\text{bits}$). For this method, we use 2×2 cores. Each core stores a copy of the temporal coding tensors and its corresponding part of the spatial coding tensor. When a timestamp arrives, its value is used to generate the K coding tensor coefficients that are added to the 16-bit compressive histograms. Each core builds its own compressive histograms and before transferring the data off-sensor the compressive histograms of the 2×2 cores are added and transferred. This implementation could be made more memory-efficient if only a single core keeps track of the compressive histograms.

For the interested reader, we have included the custom assembly implementations of the above methods in the supplementary material.

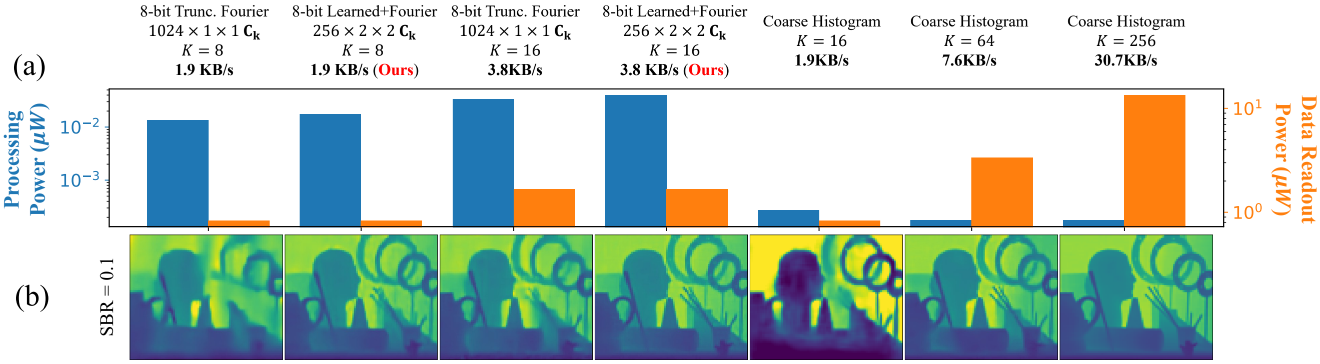
Why not use a learned $\mathbf{C}_k^{\text{temporal}}$: As shown in Suppl. Sec. S. 3.3, for certain coding tensor designs such as a separable $256 \times 2 \times 2$ $\mathbf{C}_k$, learned temporal coding tensors can outperform truncated Fourier temporal coding tensors. Unfortunately, in our current per-core implementation storing K learned separable $256 \times 2 \times 2$ coding tensors would require $(256 \cdot K + K) \cdot 8\text{bits}$ of memory per core (16,448 bits for $K = 8$). The next step in this line of work is to start adopting distributed memory implementations where the coding tensors are distributed across multiple neighboring cores to reduce the memory overhead of the coding tensors.

Processing Power Estimation: The UltraPhase chip was characterized by executing specific instructions in an infinite loop and measuring the average power consumption. The measured average power per instruction per core was 1.06e-11 Watts, and one instruction requires 3 clock cycles to execute. To estimate the average power consumption of each of the above methods, their execution time (in clock cycles) is measured, and then we estimate power consumption by assuming the worst-case scenario, when all the operations involve reading and writing to the RAM and power consumption is at maximum. Finally, given an average number of photons per depth frame, we compute the average processing power for a set of 2×2 cores/pixels that outputs 30 depth frames per second. The resulting numbers are summarized in the table on Suppl. Fig. 5.

Data Transfer/Readout Power Estimation: To estimate the data readout power, we assume a conventional digital I/O at 3.3V with a load of 5pF operating at the specified bandwidth. This is the same standard readout interface used by the SwissSPAD2 [15], which is a high-resolution 512×512 SPAD array. Given the data rate, power can be estimated as $\text{DATARATE} \cdot 5 \cdot 10^{-12} \cdot (3.3^2)$, where DATARATE is in units of bits/second.

Power Consumption Analysis: Fig. 4a shows the processing and data readout power consumption. Although compressive histogram models dissipate around 100x more power when processing compared to coarse histograms, the processing power for all methods continues to be only a small fraction of the overall power dissipated. Data readout is the dominant power consumption source, hence reducing data rates can provide higher power reductions than reducing computation.

As observed in Fig. 4b, the proposed spatio-temporal compressive histograms with $K = 16$ can provide comparable depth reconstruction quality to a 256 bin coarse histogram, while reducing data rates and consequently data readout power by 8x. To emulate 8-bit quantization for the compressive histogram models, at test time, we quantize the coding tensors to 8-bits.



Supplementary Figure 4. **UltraPhase power analysis for 2×2 pixels.** (a) Average power dissipated by 2×2 cores/pixels when processing 550 photon timestamps per pixel per depth frame at 30 frames per second. The processing power is estimated from each methods implementation on 2×2 cores of UltraPhase. The data readout power is estimated from the output data rate and assuming the same readout scheme of the SwissSPAD2 [15]. Coarse histograms have 8-bit precision, while compressive histograms have 16-bit precision, which means that a coarse histogram with twice as many coefficients will have the same data rate. The y-axis is in log scale and is displayed for the processing and readout power independently. (b) Simulated depth reconstructions for each method on a scene with an average of 550 photons detected per pixel (50 signal photons and 500 background photons). The coding tensor weights for the compressive histogram models shown here are quantized to 8-bits at test time.

Limitations of the Implementation and Analysis: For the purposes of this paper, we are able to get important insights from our implementation on UltraPhase which assumes each processing core is equivalent to a SPAD pixel. However, having access to the fine-grained spatial information of each timestamp is crucial to exploit the full potential of our proposed method. For instance, a spatio-temporal coding tensor operating on 4×4 SPAD pixels can store a compact compressive histogram that preserved spatial details, while a coarse histogram that aggregates timestamps from the 4×4 pixels will lose the spatial information.

Scaling to Megapixel SPAD Arrays: In an UltraPhase-like architecture, the amount of memory per core is unlikely to significantly increase as we scale to megapixel SPAD arrays due to pixel pitch reasons. Currently, the processing core is 107x107 microns and memory covers around 30% of that area [1]. The pixel pitch for the 4×4 SPADs connected to a core is 28 micron. However, as we scale the megapixel SPAD arrays the pixel pitch will reduce, which means the processing core area will also reduce or it will operate on a larger number of SPAD pixels. Alternatively, the processing core can reduce in size by reducing its computation capabilities (e.g., MegaPhase cores in [1] are 55x55 microns). Therefore, as we scale towards megapixel SPAD processing arrays, we can expect the shared memory across a neighborhood of SPAD pixels to be on the order UltraPhase per-core memory. This extreme memory constraint can inform and further motivate exploring more lightweight coding tensor designs and reducing their memory overhead through distributed memory implementations.

Coding Tensor Model	Compressive Histogram Length (K)	Compressive Histogram Coefficient Bit-Depth	Processing Time Per Photon [clock cycles]	Data Readout Per Pixel/Core Per Frame [Bytes]	Processing Time Per Frame with 550 Photons [seconds]	Data Rate at 30fps [KB/second]	Average Processing Power at 30fps [micro Watts]	Estimated Readout Interface Power [micro Watts]
Coarse Histogram (16 bins)	$K = 16$	8-bits	30	16	3.93e-05	1.92	2.74e-04	0.84
Coarse Histogram (64 bins)	$K = 64$	8-bits	24	64	3.14e-05	7.68	1.75e-04	3.35
Coarse Histogram (256 bins)	$K = 256$	8-bits	24	256	3.14e-05	30.72	1.75e-04	13.38
8-bit Truncated Fourier $1024 \times 1 \times 1$	$K = 8$	16-bits	210	16	2.74e-04	1.92	1.34e-02	0.84
8-bit Truncated Fourier $1024 \times 1 \times 1$	$K = 16$	16-bits	330	32	4.32e-04	3.84	3.32e-02	1.67
8-bit Learned + Fourier (**Ours**) $256 \times 2 \times 2$	$K = 8$	16-bits	241	16	3.15e-04	1.92	1.76e-02	0.84
8-bit Learned + Fourier (**Ours**) $256 \times 2 \times 2$	$K = 16$	16-bits	361	32	4.73e-04	3.84	3.98e-02	1.67

Supplementary Figure 5. **UltraPhase power analysis for 2×2 cores processing 550 photons per depth frame at 30 frames per second.** This table shows the processing time and data rates for different methods and computes the average processing power and the estimated data readout power. One clock cycle is 2.38ns in duration, and single instruction executes in 3 clock cycles. The processing time per frame is obtained by NUM_PHOTONS*PROCESSING_TIME_PER_PHOTON*2.38ns. If we operate at 30 fps, that means that the total exposure time per depth frame is 33ms, and then we are only dissipating processing power for a subset of the 33ms. Hence the average processing power at 30 fps will be given by scaling the processing power by the processing time and then dividing by 33ms.

S. 3. Supplemental Analysis of the Coding Tensor Design Space

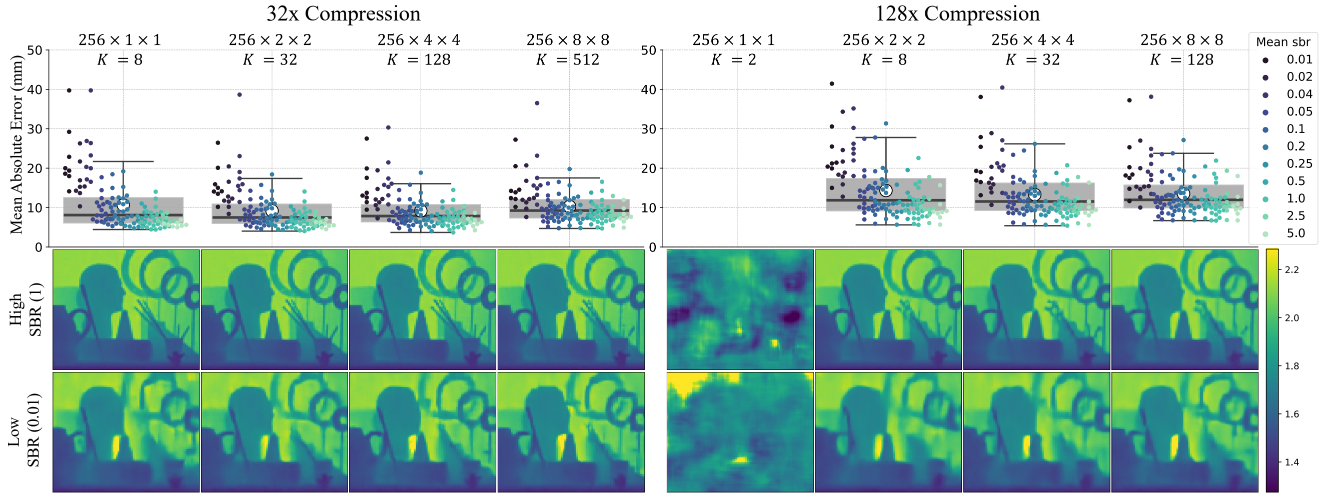
In this section, we present additional results related to the ablation study on the coding tensor design space. These results include the effect of the spatial block dimensions, the effect of the size of **C**, and the performance difference between coding tensors whose temporal dimension is learned vs coding tensors whose temporal dimension is initialized and fixed to truncated Fourier codes.

S. 3.1. When does spatio-temporal coding help?

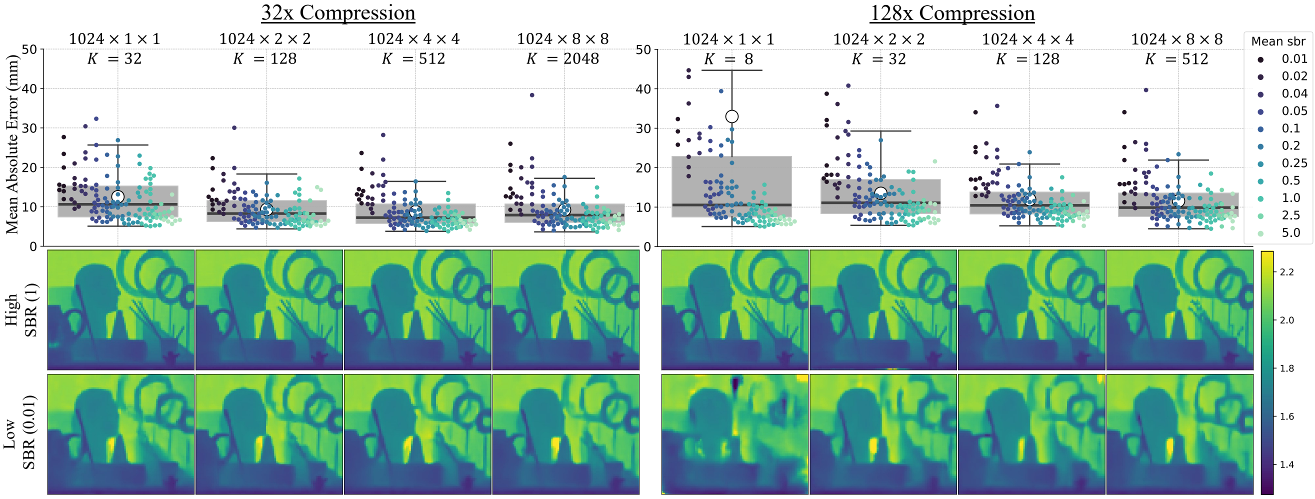
Figures 6 and 7 show the quantitative and qualitative performance of learned separable coding tensors as we vary their spatial block dimension from 1×1 up to 8×8. At low compression (32x) and high SBR levels, all models are able to produce high-quality reconstructions that recover both coarse and fine details of the scene. At low compression (32x) and extremely low SBR, models with a spatial block larger than 1×1 are able to better preserve some of the coarser scene details such as the sticks. However, quantitatively, the overall performance difference is small. At high compression and high SBR, models with a 2×2 and 4×4 spatial block are able to recover fine details that are blurred in the 8×8, which are particularly noticeable in Fig. 7. Finally, in the most challenging scenario where we have high compression and low SBR, it becomes essential to use a coding tensor that aggregates information from neighboring spatial locations. In this scenario, although all methods blur some fine scene details, the coding tensors with large spatial dimensions better preserve coarser details such as the pot handle and the sticks.

Why does $256 \times 1 \times 1$ fail at 128x compression? As observed in Fig. 6, the coding tensor with dimensions $256 \times 1 \times 1$ fails when only $K = 2$ coding tensors are learned. Although, it is possible to reconstruct depths from a compressive histogram with as few as 2 coded projections [6], finding two coding tensors that can lead to an unambiguous depth range without further regularization can be challenging.

Summary: Overall, coding tensors that exploit spatial correlations are more robust to low SBR settings. However, at high SBR, operating in a large spatial neighborhood can make it harder to resolve fine scene details. Moreover, increasing the spatial block dimension further increases the number of parameters of the coding tensor which, as discussed in the main paper, is less practical. In this analysis, we find that coding tensors with spatial block of 2×2 and 4×4 achieve good balance of robustness to noise at low SBR, while being able to reconstruct fine scene details at high SBR.



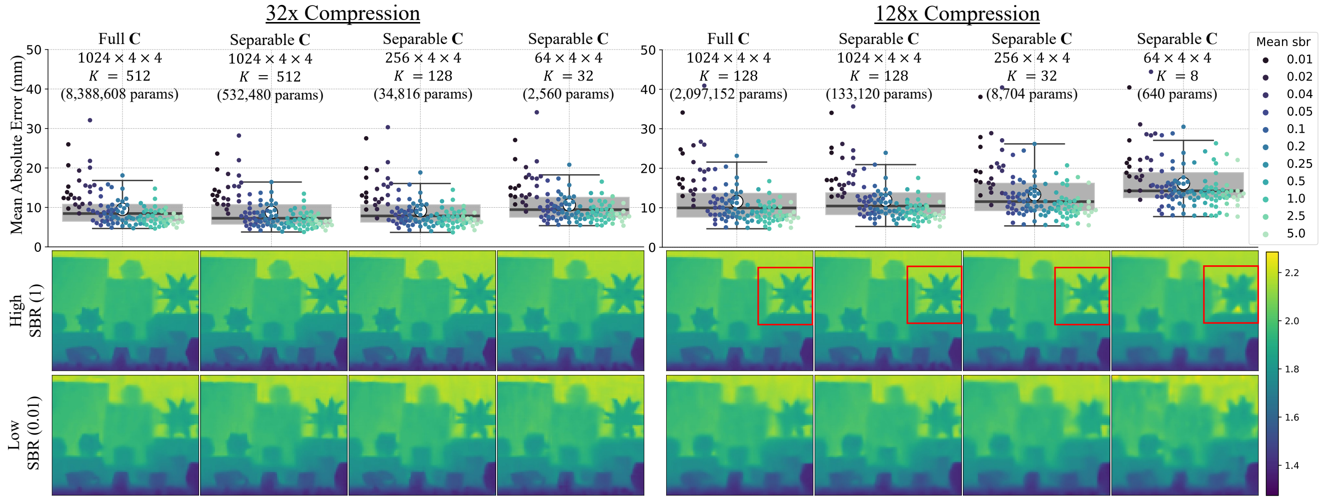
Supplementary Figure 6. **Effect of Spatial Tensor Dimension for** $M_t = 256$ Each point in the scatter plots show the MAE for a given test scene and their color hue represent the mean SBR level used in that simulation. The horizontal black line, white circle, gray box, and error bars correspond to the median, mean, quartiles, and 1.5x the inter-quantile range, respectively. Outliers with an MAE larger than 50mm are not visible in the plot, however, they are included in the calculation of the statistics. The images directly below each model correspond to the depth reconstructions for two test examples at low and high SBR levels whose mean signal and background photon detections per pixel are [10, 10] and [10, 1000], respectively. For a fixed compression level, the spatial block size of each model is increased from left to right and K is adjusted to maintain the same compression level. The coding tensors for all models in this plot are learned and separable for spatial blocks larger than 1×1.



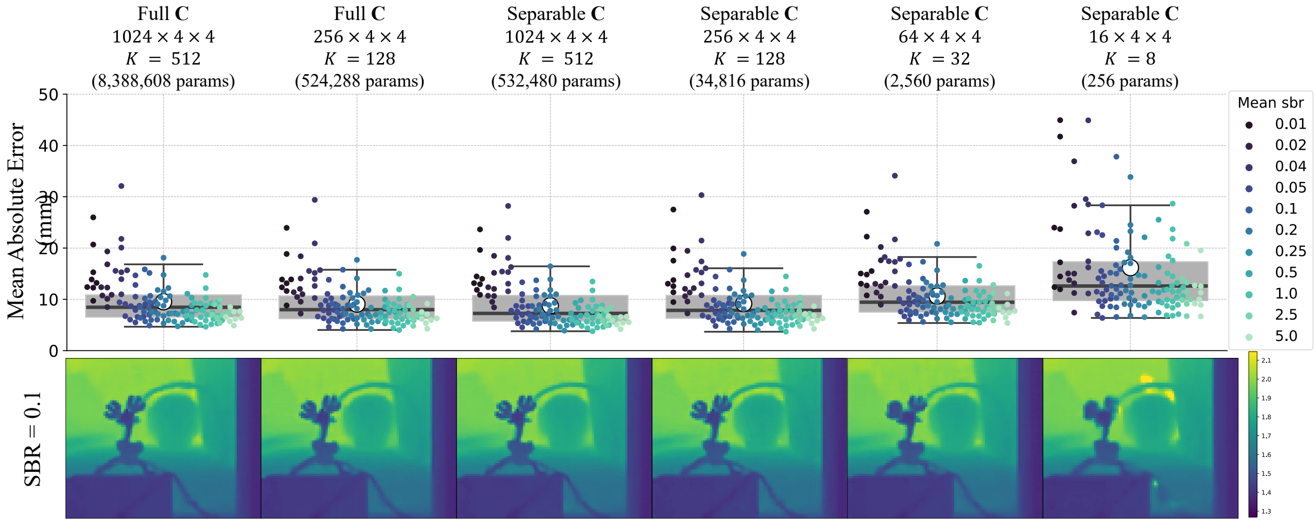
Supplementary Figure 7. **Effect of Spatial Tensor Dimension for $M_t = 1024$** Each point in the scatter plots show the MAE for a given test scene and their color hue represent the mean SBR level used in that simulation. The horizontal black line, white circle, gray box, and error bars correspond to the median, mean, quartiles, and 1.5x the inter-quantile range, respectively. Outliers with an MAE larger than 50mm are not visible in the plot, however, they are included in the calculation of the statistics. The images directly below each model correspond to the depth reconstructions for two test examples at low and high SBR levels whose mean signal and background photon detections per pixel are [10, 10] and [10, 1000], respectively. For a fixed compression level, the spatial block size of each model is increased from left to right and K is adjusted to maintain the same compression level. The coding tensors for all models in this plot are learned and separable for spatial blocks larger than 1×1.

S. 3.2. How does reducing the size of the coding tensor affect accuracy?

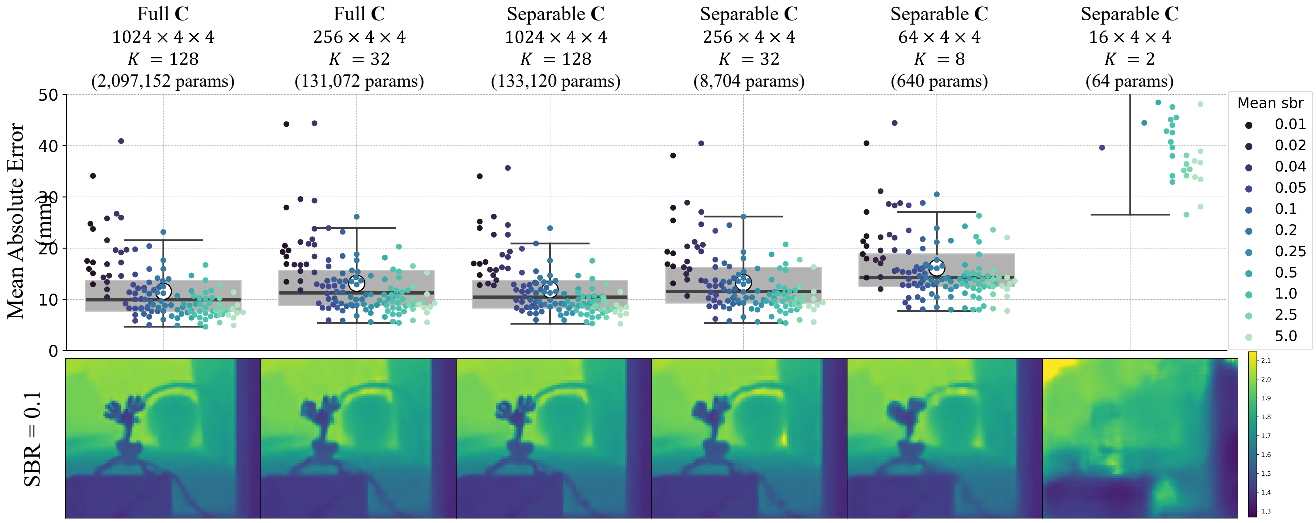
Figures 8, 9, and 10 show the quantitative and qualitative performance of different learned coding tensors as we reduce the number of parameters from left to right. The number of parameters is reduced by either making the coding tensors separable or making their temporal dimension smaller. At low compression levels (32x compression), coding tensors with as few as 2,560 parameters perform comparably to larger coding tensors with 100x more parameters. At higher compression levels (128x compression), the size of the coding tensors starts having a more pronounced effect on depth image quality. At higher SBR levels (i.e., $SBR \geq 0.1$), the larger coding tensors are able to better recover fine scene structures such as the spikes in Fig. 8 or the toy reindeer antlers in 10. Nonetheless, coding tensors with as few as 8,704 parameters can continue to perform comparably to coding tensors with millions of parameters.



Supplementary Figure 8. **How does Reducing Size of C Affect Performance?.** Each point in the scatter plots show the MAE for a given test scene and their color hue represent the mean SBR used in that simulation. The horizontal black line, white circle, gray box, and error bars correspond to the median, mean, quartiles, and 1.5x the inter-quantile range, respectively. Outliers with an MAE larger than 50mm are not visible in the plot, however, they are included in the calculation of the statistics. The images directly below each model correspond to the depth reconstructions for two test examples at low and high SBR levels whose mean signal and background photons per pixel are [10, 10] and [10, 1000], respectively. For a fixed compression level, the size of the coding tensors is reduced from left to right by making the coding tensors separable and also reducing the temporal dimension. All coding tensors are learned.



Supplementary Figure 9. **How does Reducing Size of C Affect Performance at 32x Compression?** Each scatter plot point corresponds to the MAE for each test scene. The depth images directly below each model correspond to the reconstruction of one test scene whose mean signal and background photons per pixel are [50, 500]. The size of the coding tensors is reduced from left to right by making the coding tensors separable and also reducing the temporal dimension.



Supplementary Figure 10. **How does Reducing Size of C Affect Performance at 128x Compression?** Each scatter plot point corresponds to the MAE for each test scene. The depth images directly below each model correspond to the reconstruction of one test scene whose mean signal and background photons per pixel are [50, 500]. The size of the coding tensors is reduced from left to right by making the coding tensors separable and also reducing the temporal dimension.

S. 3.3. Learned vs. Fourier-based Temporal Compressive Representations

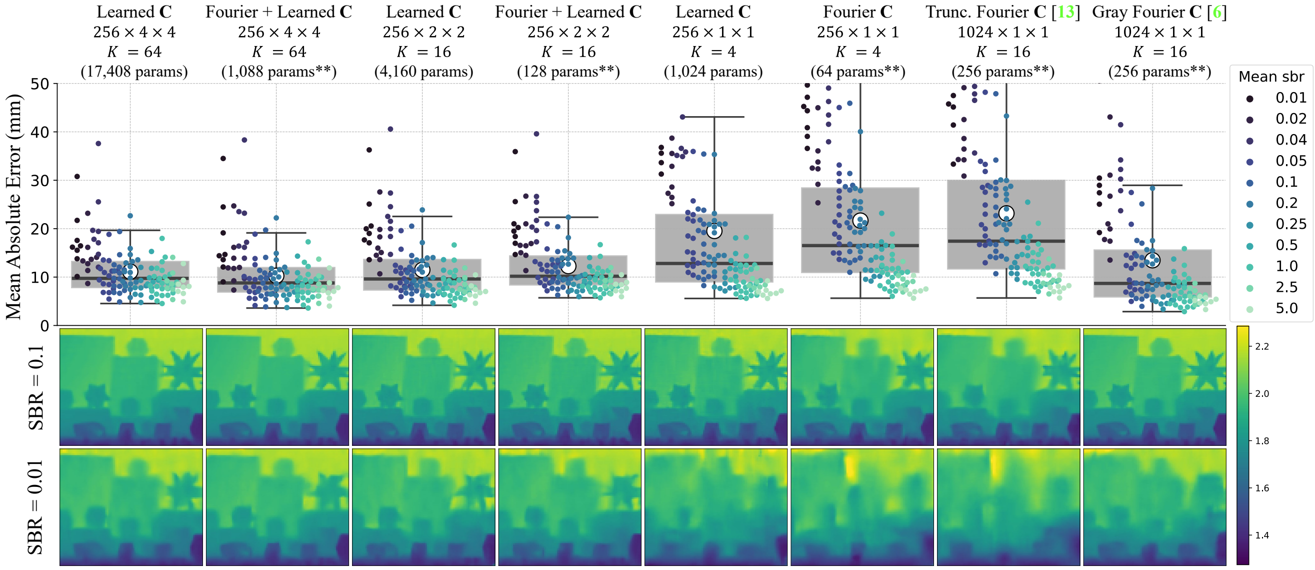
In this section, we compare the performance of separable coding tensors whose $\mathbf{C}_k^{\text{temporal}}$ is either learned or fixed to a truncated Fourier coding tensor during training.

Temporal Gray Fourier C [6]: In addition to comparing with the Truncated Fourier coding tensors proposed in [13], we also compare against another Fourier-based coding tensor design proposed in [6], which we refer to as Gray Fourier. A Gray Fourier compressive histogram uses coding tensors with dimensions $1024 \times 1 \times 1$. The coding tensors are a Fourier matrix where every two rows the frequency of the sinusoidal signal doubles as illustrated in Suppl. Fig. 17. Similar to our proposed approach, this $\mathbf{C}$ is implemented as a compressive histogram layer, with fixed weights, whose outputs are processed by the depth estimation 3D CNN.

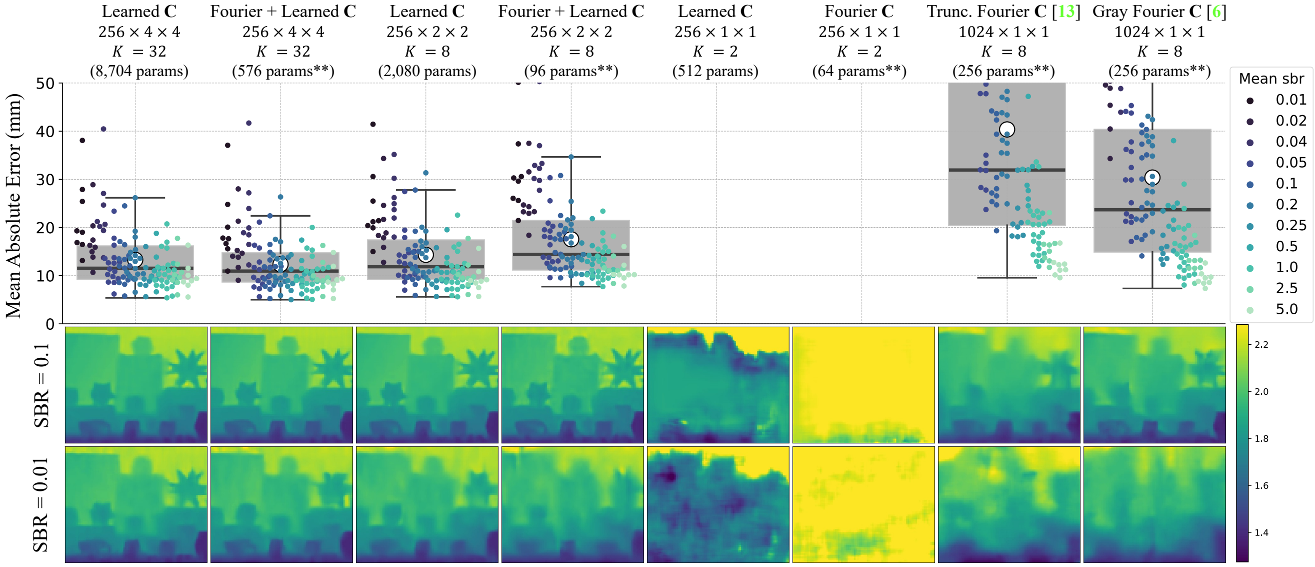
Fourier + Learned C: In this separable coding tensor design, the temporal coding tensors ($\mathbf{C}_k^{\text{temporal}}$) are fixed to truncated Fourier coding tensors, and the spatial coding tensors ($\mathbf{C}_k^{\text{spatial}}$) are learned. The temporal coding tensors in this design can be represented with a small number of parameters that do not scale with K as discussed in Suppl. Sec. S. 2, hence, the in-sensor memory overhead they introduce is smaller than a fully learned coding tensor.

Results: Supplementary Figures 11 and 12 show the overall test set performance and qualitative depth reconstructions for multiple compressive histogram models at 64x and 128x compression, respectively. As discussed in previous sections, coding tensors that exploit spatial information (e.g., $256 \times 4 \times 4$ or $256 \times 2 \times 2$) provide higher quality reconstructions, especially, at lower SBR levels. At 64x compression (Suppl. Fig. 11), models with Fourier or learned $\mathbf{C}_k^{\text{temporal}}$ perform comparably at all SBR levels. At 128x compression (Suppl. Fig. 12), a fully learned coding tensor with dimensions $256 \times 2 \times 2$ can provide some performance improvements for low SBR scenes over the Fourier + Learned $256 \times 2 \times 2$ coding tensor. Nonetheless, at 128x compression, the $256 \times 4 \times 4$ coding tensors provide the best performance.

Summary: Fourier-based temporal coding tensors have a memory-efficient implementation that does not scale with K. Using our flexible spatio-temporal compressive histogram framework, we can design coding tensors whose temporal dimension is fixed to Fourier codes and the spatial coding tensors are learned. This results in a practical compressive histogram model that can be implemented in existing SPAD pixels as shown in Suppl. Sec. S. 2 while providing robust performance across SBR and compression levels. Fully learned coding tensors can still provide some improvements in the most challenging situations (high compression and low SBR), however, they require additional in-sensor memory. Nonetheless, this additional in-sensor memory may be negligible in implementations where a large number of SPAD pixels share the same copy of the coding tensors.



Supplementary Figure 11. **Learned vs. Fourier Temporal Coding Tensors at 64x Compression.** Each scatter plot point corresponds to the MAE for each test scene. The depth images directly below each model correspond to the reconstruction of one test scene whose mean signal and background photons per pixel are [50, 500] and [10, 1000]. All models use separable coding tensors. **The number of parameters for all coding tensors based on Fourier codes is calculated assuming the memory efficient representation described in Suppl. Sec. S. 2.



Supplementary Figure 12. **Learned vs. Fourier Temporal Coding Tensors at 128x Compression.** Each scatter plot point corresponds to the MAE for each test scene. The depth images directly below each model correspond to the reconstruction of one test scene whose mean signal and background photons per pixel are [50, 500] and [10, 1000]. All models use separable coding tensors. **The number of parameters for all coding tensors based on Fourier codes is calculated assuming the memory efficient representation described in Suppl. Sec. S. 2. The models trained with $256 \times 1 \times 1$ coding tensors at this compression level are not able to converge and are not able to learn how to reconstruct the scene. This is likely due to the fact that only $K = 2$ coding tensors are used, which as discussed in Suppl. Sec. S. 3.1, can make the optimization challenging.

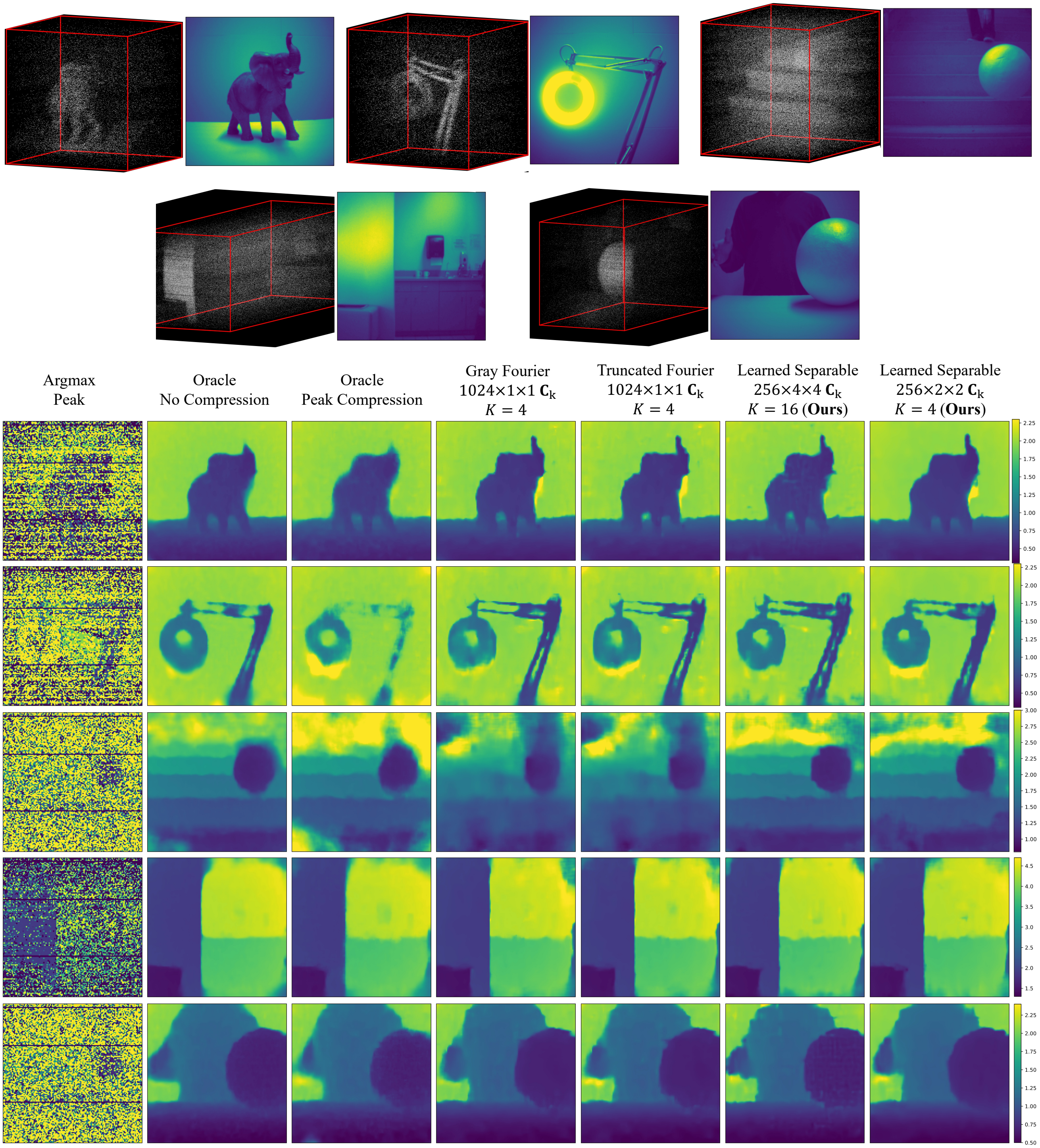
S. 4. Evaluation on Real-world Data

To evaluate the generalization of the proposed models, we downloaded raw histogram tensor data captured by [9] with a SPAD-based 3D camera prototype. The dataset was captured with a line scanning system composed of a co-located picosecond laser and a 1D LinoSPAD array with 256 SPAD pixels [3]. The histogram tensors have $N_t = 1536$ time bins, a spatial resolution of 256×256, and a bin size $\Delta = 26$ps. The raw histogram tensors are downsampled to be $1024 \times 128 \times 128$ to make the time domain compatible with the learned coding tensors that use $M_t = 1024$ and also to avoid out-of-memory errors.

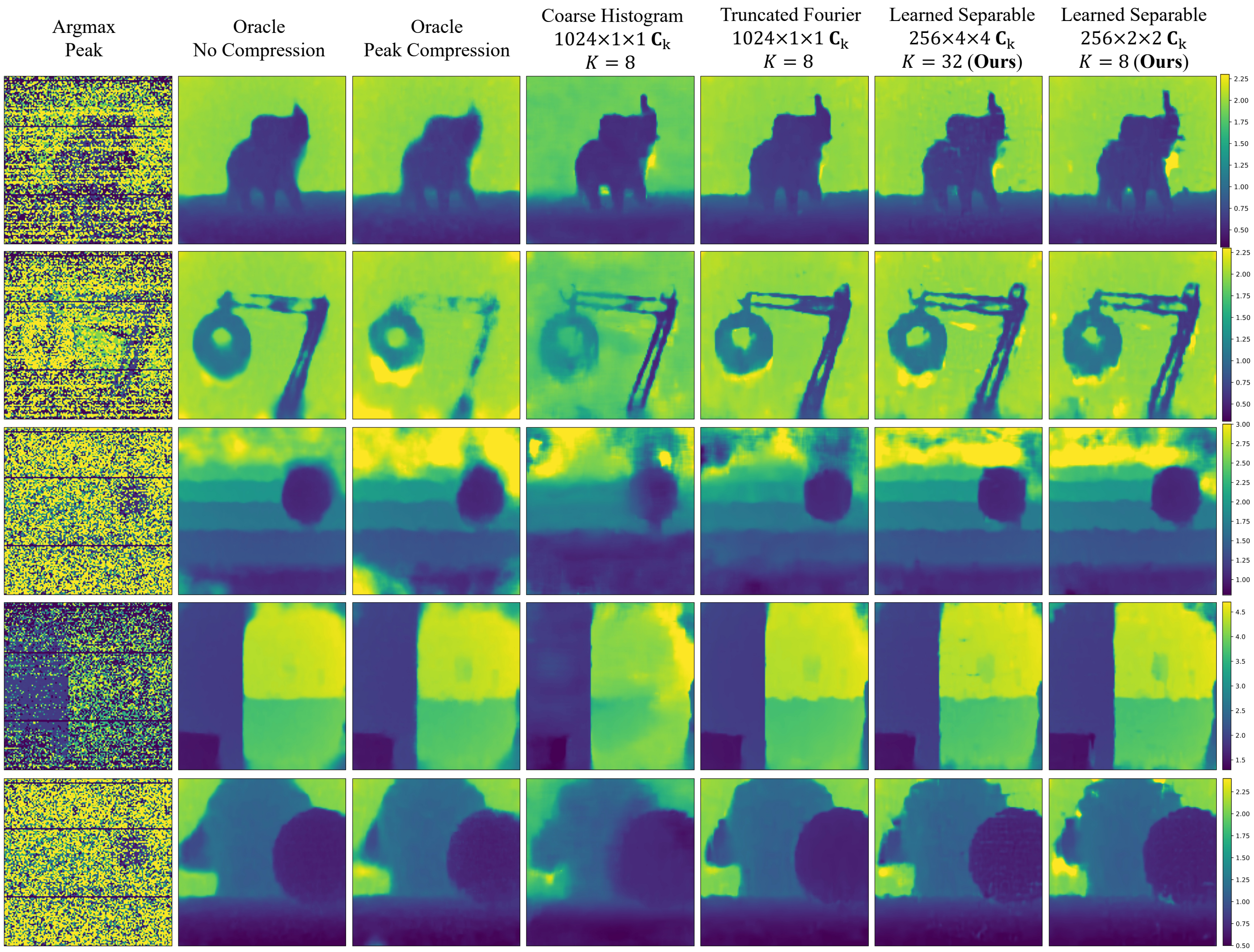
Fig. 13 and 14 show the depth reconstructions for the oracle baselines and multiple compressive histograms at 256x and 128x compression, respectively. All models are able to produce plausible depth reconstructions, suggesting good generalization to real-world data. However, all compressive histogram models display small artifacts throughout the image that could be due to high noise levels or generalization problems. These artifacts seem to be avoided by the oracle baselines (no compression and peak compression) by over-smoothing the images. This over-smoothing is due to the total variation regularizer that we used for the oracle baselines but not for the compressive histogram models, which we found produced the better oracle models on the synthetic datasets. Therefore, these results suggest that a spatial regularizer can be used to improve the compressive histogram model's generalization on real-world data. Nonetheless, despite these minor artifacts, the depth reconstructions suggest good generalization by all models to these challenging scenarios.

Comparison with Fourier-based $\mathbf{C}$: We observe that the models that used a Gray-based Fourier or a Truncated Fourier $\mathbf{C}$ produced blurrier depth reconstructions than the learned $\mathbf{C}$ models. This can be observed at 128x compression in the lamp scene where the wires merge into a single blob, or in the staircase scene where the stair edges are blurred. At 256x compression, the blurring in the staircase scene is even more significant, likely due to low SBR since the scene is outdoors. On the other hand, the models with a learned $\mathbf{C}$ produce sharper depth reconstructions at the same compression level, despite being trained in the exact same manner.

Comparison with Coarse Histogramming: Although, the depth images for the coarse histogramming coding tensor shown in Fig. 14 look reasonable qualitatively, they have large absolute depth errors when comparing them to the other approaches. A coarse histogram will often produce a quantized depth image [13, 6], however, the depth estimation 3D CNN learns to smooth and upsample the coarse histogram and produce more plausible depth images. Nonetheless, coarse histograms consistently produce less accurate depth reconstructions than other compressive histogram approaches.



Supplementary Figure 13. **Depth Reconstructions of Real-world SPAD Data at 256x Compression.** Depth reconstructions of different scenes captured with a SPAD-based 3D camera prototype [9]. The first two rows show a high-resolution intensity image of the captured scene and a point cloud visualization of the raw histogram tensor of that same scene. Gray Fourier and Truncated Fourier correspond to the compressive histogram methods proposed in [6] and [13], respectively. For more details on these methods see Suppl Sec. S. 3.3.


Argmax Peak

Oracle No Compression

Oracle Peak Compression

Coarse Histogram $1024\times1\times1\ \mathbf{C}_\mathrm{k}$ $K = 8$

Truncated Fourier $1024\times1\times1\ \mathbf{C}_\mathrm{k}$ $K = 8$

Learned Separable $256\times4\times4\ \mathbf{C}_\mathrm{k}$ $K = 32$ (**Ours**)

Learned Separable $256\times2\times2\ \mathbf{C}_\mathrm{k}$ $K = 8$ (**Ours**)

Supplementary Figure 14. **Depth Reconstructions of Real-world SPAD Data at 128x Compression.** Depth reconstructions of different scenes captured with a SPAD-based 3D camera prototype [9]. The two rows show a high-resolution intensity image of the captured scene and a point cloud visualization of the raw histogram tensor of that same scene.

S. 5. Further Datasets and Implementation Details

S. 5.1. Simulating SPAD Measurements

In this section, we provide a detailed description of how SPAD measurements are simulated for the synthetic datasets used in this paper.

Given an RGB-D image, pulse waveform ($h(t)$), and the mean number of detected signal ($\Phi^{\text{sig}}_{\text{mean}}$) and background ($\Phi^{\text{bkg}}_{\text{mean}}$) photons per pixel, we set the photon detection parameters for Eq. 2 as follows. First, we calculate the amplitude of the illumination signal arriving at each pixel ($a_{\boldsymbol{p}}$ in Eq. 1) by using the reflectance at that pixel and accounting for the intensity radial fall-off due to distance. Similar to [9, 12], the NYUv2 training set reflectance was estimated using intrinsic image decomposition on the blue channel of the RGB image, and the Middlebury testing set reflectance was estimated using the mean of the RGB channels. Intrinsic image decomposition can lead to more accurate reflectance estimates for non-lambertian surfaces. Consequently, given $a_{\boldsymbol{p}}$, $h(t)$, the per-pixel depths, and the average number of signal photon per pixel, we can scale the average number of signal photons arriving at each time bin such that $\sum_{\boldsymbol{p}} \sum_i \Phi^{\text{sig}}_{i,\boldsymbol{p}} = \Phi^{\text{sig}}_{\text{mean}}$. Similarly, we can emulate the per-pixel background illumination (Φ^{bkg}) using the RGB channel mean and scaling it such that it matches the desired mean number of background photons per pixel. Finally, we can add dark counts to the per-pixel background illumination component. This step is only done on the training set using a calibration dark count image obtained from the hardware prototype in [9]. We observe that the models trained with this dark count component generalize well to histogram tensors without the dark counts, as shown in our test results.

Summary: Overall, the SPAD measurement simulation pipeline used in this paper is the one originally developed by [9], and later used in [12, 14]. The only significant difference is that our simulated test set includes additional mean signal and background photon count settings.

S. 5.2. Training and Implementation Details

In this section, we provide further training and implementation details.

All models in this paper are implemented in PyTorch [11]. The input to all the models is a 3D histogram tensor. Recall that due to the linearity of compressive histograms, encoding the histogram tensors is equivalent to encoding each individual photon timestamp and summing them up. Hence, models deployed with a compressive histogram layer can also take as input a stream of photon timestamps and build the compressive histogram.

Compressive Histogram Layer: The compressive histogram layer is implemented as a single-layer encoder and decoder. The encoder is a 3D convolution with a stride equal to the filter size. The coding tensors, $\mathbf{C}_k$, are the learned filters. All coding tensors are constrained to be zero-mean along the time dimension. This constraint makes the expected encoded value for background photons distributed uniformly along the time dimension be 0 [13]. The outputs of the encoder are the compressive histograms $\widehat{\boldsymbol{Y}}_b$. The decoder is an unfiltered backprojection that is implemented as a 3D transposed convolution with a stride equal to its filter size.To help the CNN model generalize to different photon count levels we apply zero-normalization along the channel dimension (i.e., K) to the inputs ($\widehat{\boldsymbol{Y}}_b$) and the weights ($\mathbf{C}$) of the transposed convolution as follows:

$$\text{ZN}(\widehat{\boldsymbol{Y}}_b) = \frac{\widehat{\boldsymbol{Y}}_b - \mathbb{E}(\widehat{\boldsymbol{Y}}_b)}{\left\|\widehat{\boldsymbol{Y}}_b - \mathbb{E}(\widehat{\boldsymbol{Y}}_b)\right\|_2}, \quad \text{ZN}(\mathbf{C}) = \frac{\mathbf{C} - \mathbb{E}(\mathbf{C})}{\|\mathbf{C} - \mathbb{E}\left(\mathbf{C}\right)\|_2} \tag{1}$$

where the mean and L2 norm are computed over the channel dimension. This normalization is also known as layer normalization [2]. Eq. 1 is inspired by the zero-normalized cross-correlation depth decoding algorithm used in ToF imaging [7, 5, 6] and structured light [10, 4].

Depth Estimation 3D CNN Model: To estimate depths from the decoded histogram tensor we use the 3D deep boosting CNN model proposed by [12] for single-photon 3D imaging. Different from [12], our implementation does not include a non-local block after the feature extraction stage. The output of the model is a denoised histogram tensor, $\boldsymbol{H}^{\text{out}}$, from which depths are estimated using a softargmax function along the time dimension. Similar to [9, 12] we use the pixel-wise Kullback-Leibler (KL) divergence between the denoised histogram tensor and a normalized ground truth histogram tensor, $\boldsymbol{H}^{\text{gt}}$, as our objective function. This loss can be written for each pixel, $\boldsymbol{p}$, as:

$$L_{\text{KL}}(\boldsymbol{H}^{\text{gt}}_{\boldsymbol{p}}, \boldsymbol{H}^{\text{out}}_{\boldsymbol{p}}) = \sum_{i=0}^{N_t-1} \boldsymbol{H}^{\text{gt}}_{i,\boldsymbol{p}} \log\left(\frac{\boldsymbol{H}^{\text{gt}}_{i,\boldsymbol{p}}}{\boldsymbol{H}^{\text{out}}_{i,\boldsymbol{p}}}\right) \tag{2}$$

Training: At each training iteration we randomly sample patches of size $1024 \times 32 \times 32$ from the training set. We train all models using the ADAM optimizer [8] with default parameters ($\beta_1 = 0.9, \beta_2 = 0.999$), batch size of 4, and an initial learning rate of 0.001 that decays by 0.9 after every epoch. We train all models for 30 epochs with checkpoints every half an epoch, and for a given model we choose the checkpoint that achieves the lowest root mean squared error (RMSE) on the validation set.

S. 6. Analysis of the Memory Overhead of Coding Tensors

Compressive histograms have the potential to greatly reduce off-sensor data transmissions and the amount of in-sensor memory required compared to a conventional histogram tensor representation. However, the general compression framework introduced in Section 4 requires the in-sensor storage of the K coding tensors ($\mathbf{C} = (\mathbf{C}_k)_{k=0}^{K-1}$) that are used for compression. This means that a large $\mathbf{C}$ may introduce a significant amount of in-sensor memory overhead, making these designs for $\mathbf{C}$ less practical. In this section, we provide a quantitative analysis of this memory overhead for different coding tensor designs.

Recall that $\boldsymbol{H}$ and $\mathbf{C}$ are $N_t \times N_r \times N_c$ and $K \times M_t \times M_r \times M_c$ tensors, respectively. Let, $N = N_t \cdot N_r \cdot N_c$ be the total number of elements in the histogram tensor. Moreover, let $M = M_t \cdot M_r \cdot M_c$ the size of a single coding tensor which is also the size of the histogram block, $\boldsymbol{H}_b$ that we are compressing. For the remainder of this analysis, we assume the following:

1. We assume that all histogram blocks $\boldsymbol{H}_b$ that are compressed are non-overlapping. This means that the total number of compressive histograms that are transferred off-sensor is $B = N/M$.
2. We assume that only a single $\mathbf{C}$ is stored inside the sensor. This $\mathbf{C}$ will be shared among all SPAD pixels.
3. We assume that the elements of $\mathbf{C}$ and $\boldsymbol{H}$ are represented using the same number of bits.

Table 1 provides the expected compression ratios for off-sensor data transmission and in-sensor storage. These two compression ratios will differ due to the memory overhead incurred by compressive histograms when having to store the coding tensors $\mathbf{C}$. It is clear that a compressive histogram for a histogram block whose size equals the size of the histogram tensor (i.e., $M = N$), would actually require more in-sensor memory than the histogram tensor making this compressive histogram design impractical.

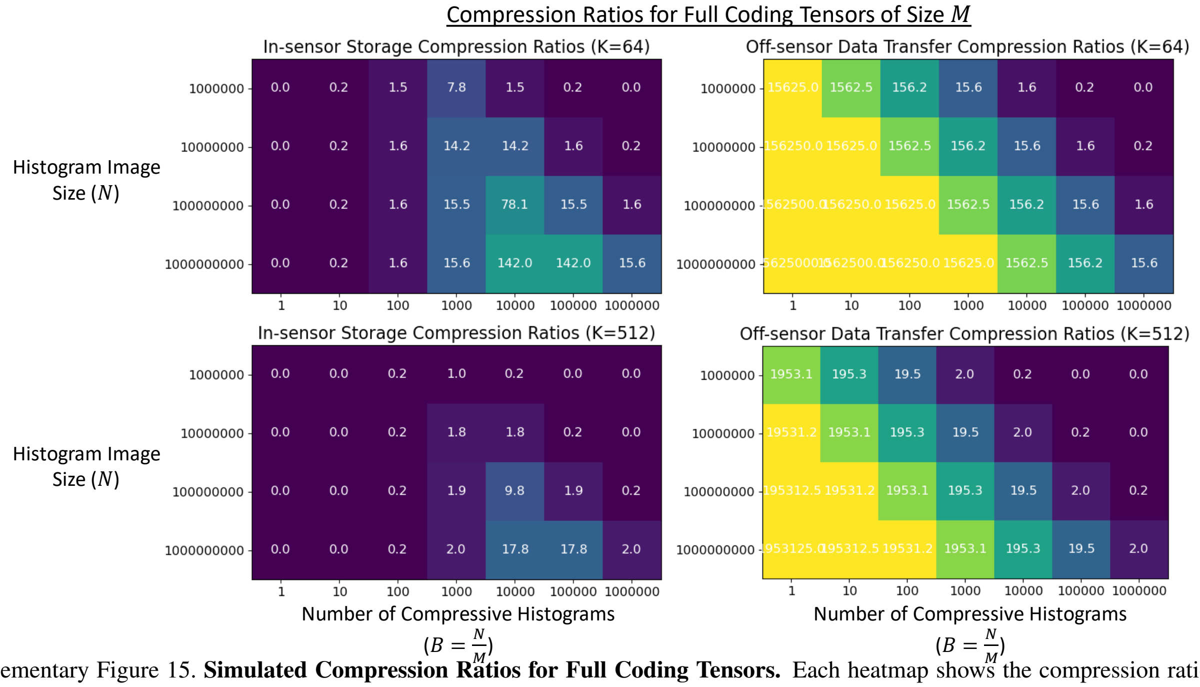
	Histogram Tensor	**Compressive Histograms**	**Compression Ratios**
Off-sensor Data Transmission	N	$B \cdot K$	$N/(B \cdot K)$
In-sensor Storage	N	$(K \cdot B) + (K \cdot M) = K \cdot (B + M)$	$N/(K \cdot (B + M))$

Table 1. **Data Transmission and In-sensor Storage Requirements.** This table shows the off-sensor data transmission and in-sensor storage requirements for a histogram tensor of size N and a set of B compressive histograms that use K coding tensors of size M for compression. The compression ratio column shows the amount of compression that can be achieved for off-sensor data transmission and in-sensor storage.

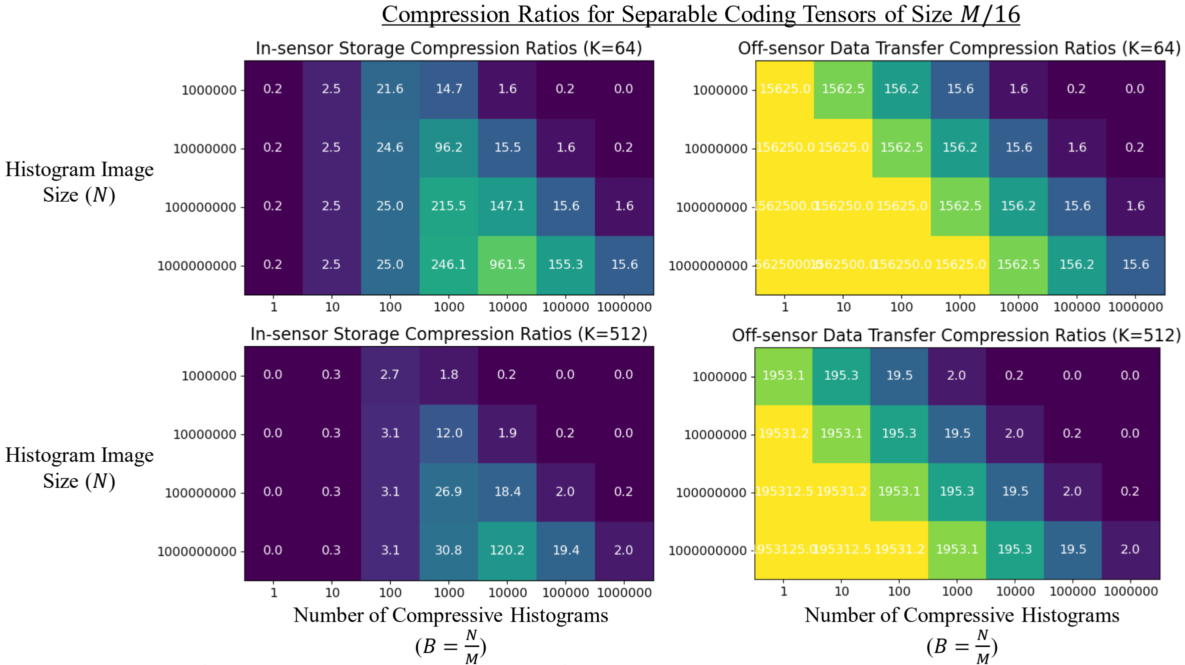
Compression Ratios for *Full* Coding Tensors: Fig. 15 shows the expected compression ratios for different histogram tensor and coding tensor sizes. As we reduce the number of compressive histograms to represent the histogram tensor, the size of the coding tensors will increase and consequently we achieve lower in-sensor compression. Since the coding tensors do not need to be transferred off-sensor, the data rate compression ratio continue to increase as we reduce the number of compressive histograms because the overall size of the compressive representation does decrease when K is fixed. Overall, a good balance between reducing in-sensor memory and data transmission seems to be achieved when using 10,000-100,000 compressive histograms to represent a histogram tensor with 1e9 element (e.g., a 1 megapixel SPAD array with 1000 bins per pixel). In this case, the size of a single coding tensor (M) should range between $10,000 - 100,000$ for $64 \leq K \leq 512$. Some of the coding tensors with $K \geq 64$ that were evaluated in this paper approximately match this size range, e.g., $M = 16,384$ for $1024 \times 4 \times 4$ or for $256 \times 8 \times 8$.

Compression Ratios for *Separable* Coding Tensors: Fig. 16 shows the expected compression ratios for different histogram tensor and coding tensor sizes for a separable coding tensor that is 16x smaller than a full coding tensor. A separable coding tensor that is $\sim$ 16x smaller is consistent with the separable coding tensors used in the main paper. For instance, a $256 \times 4 \times 4$ $\mathbf{C}_k$ is $\sim$ 16x smaller if we make its temporal and spatial dimensions separable. In this scenario, a good balance between in-sensor storage and data transmission compression is achieved when using 1,000-100,000 compressive histograms to represent a histogram tensor with 1e9 elements. In this case, the size of a single separable coding tensor should range between $625 - 62,500$ for $64 \leq K \leq 512$. Some of the separable coding tensors with $K \geq 64$ that were evaluated in this paper approximately match this size range: $M = 272$ ($256 \times 4 \times 4$), $M = 1040$ ($1024 \times 4 \times 4$).

Summary: Parameter-efficient coding tensors can reduce the in-sensor memory overhead that compressive histograms introduce. In this section, we show that the local block-based separable coding tensor designs explored in this paper are able to reduce the memory overhead for histogram tensors of size $N \geq$ 1e7. Additional lightweight $\mathbf{C}$ designs could rely on other factorization techniques such as low-rank approximations. Moreover, as shown in Suppl. Sec. S. 2, weight quantization can be an extremely effective technique in further compressing $\mathbf{C}$. Finally, $\mathbf{C}$ designs whose parameters can be computed on the fly, such as Fourier-based (Suppl. Sec. S. 2) or Gray codes [6], are a practical design when multiple $\mathbf{C}$ need to be stored across the SPAD array. Ultimately, a practical coding tensor representation will be determined by the hardware constraints of a given SPAD camera.

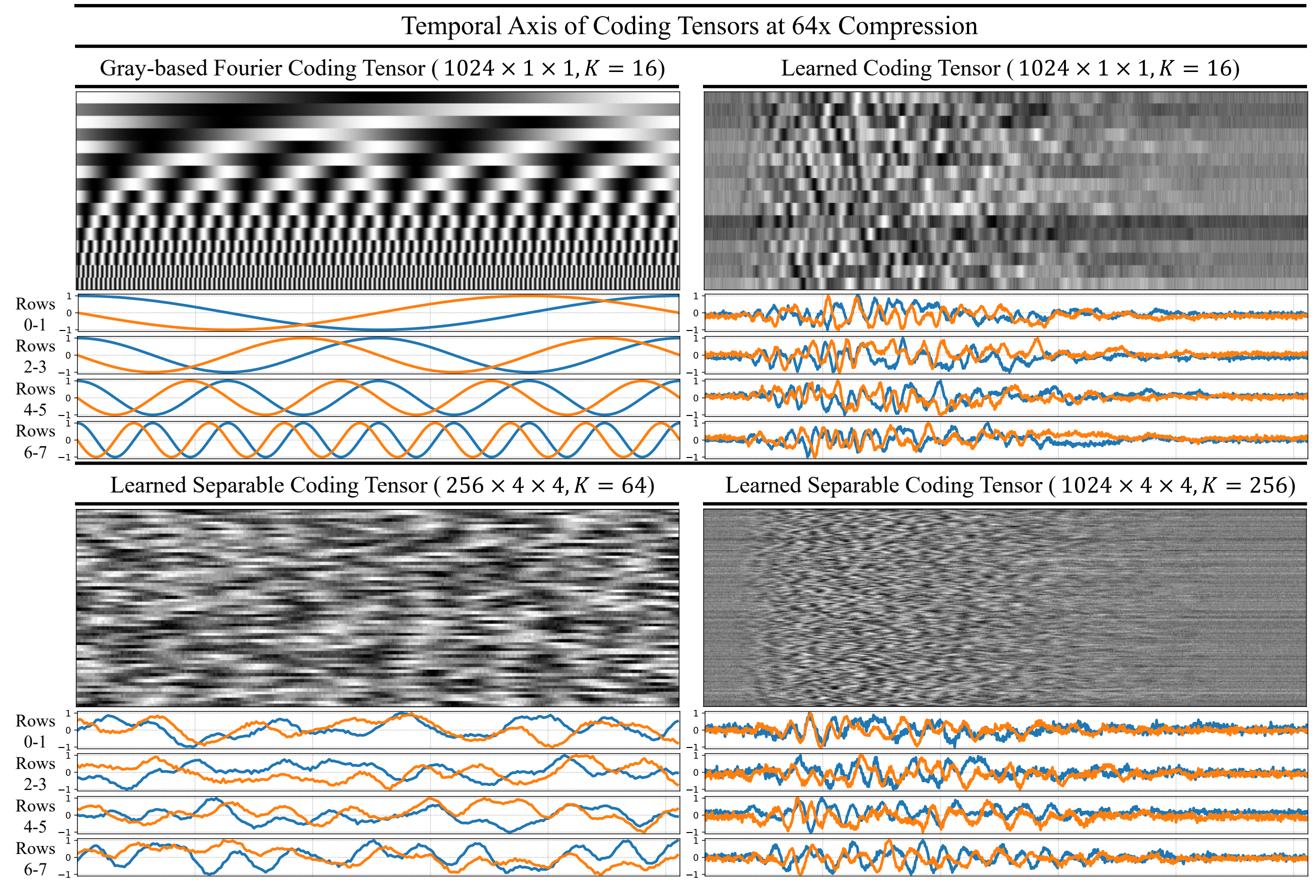


Supplementary Figure 15. **Simulated Compression Ratios for Full Coding Tensors.** Each heatmap shows the compression ratio for a fixed K for different histogram tensor sizes (B) and number of compressive histograms (B) that are used. The compression ratios for in-sensor storage (left column) and data transfer (right column) are computed using the equations in Table 1.



Supplementary Figure 16. **Simulated Compression Ratios for Separable Coding Tensors.** Each heatmap shows the compression ratio for a fixed K for different histogram tensor sizes (B) and number of compressive histograms (B) that are used. We assume that a separable coding tensor is 16x smaller than a full coding tensor, which is consistent with the separable coding tensors used in the paper. The compression ratios for in-sensor storage and data transfer are computed using the equations in Table 1 replacing M with $M/16$.

S. 7. Additional Coding Tensor Visualization


Temporal Axis of Coding Tensors at 64x Compression

Gray-based Fourier Coding Tensor ($1024 \times 1 \times 1, K = 16$)

Learned Coding Tensor ($1024 \times 1 \times 1, K = 16$)

Learned Separable Coding Tensor ($256 \times 4 \times 4, K = 64$)

Learned Separable Coding Tensor ($1024 \times 4 \times 4, K = 256$)

Supplementary Figure 17. **Temporal Dimension of C at 64x Compression** Visualization of the temporal dimension for different coding tensors that achieve 64x compression. The matrix visualized for coding tensors of dimension $1024 \times 1 \times 1$ (columns 1 and 2) is an 16×1024 matrix, since $K = 16$ and $M_t = 1024$. On the other hand, the matrix for the learned separable $256 \times 4 \times 4$ $\mathbf{C}_k$ is 64×256 since $K = 64$ and $M_t = 256$. Similarly, the matrix for the learned separable $1024 \times 4 \times 4$ $\mathbf{C}_k$ is a 256×1024 matrix. The functions shown below each matrix correspond to different rows of the matrix. The Gray Fourier matrix corresponds to the compressive histogram method proposed in [6].

References

[1] Andrei Ardelean. Computational imaging spad cameras. page 164, 2023. 4, 5

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 16

[3] Samuel Burri, Claudio Bruschini, and Edoardo Charbon. Linospad: a compact linear spad camera system with 64 fpga-based tdc modules for versatile 50 ps resolution time-resolved imaging. *Instruments*, 1(1):6, 2017. 13

[4] Wenzheng Chen, Parsa Mirdehghan, Sanja Fidler, and Kiriakos N Kutulakos. Auto-tuning structured light by optical stochastic gradient descent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5970–5980, 2020. 16

[5] Felipe Gutierrez-Barragan, Huaijin Chen, Mohit Gupta, Andreas Velten, and Jinwei Gu. itof2dtof: A robust and flexible representation for data-driven time-of-flight imaging. *arXiv preprint arXiv:2103.07087*, 2021. 16

[6] Felipe Gutierrez-Barragan, Atul Ingle, Trevor Seets, Mohit Gupta, and Andreas Velten. Compressive single-photon 3d cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17854–17864, 2022. 1, 4, 7, 11, 13, 14, 16, 18, 20

[7] Felipe Gutierrez-Barragan, Syed Azer Reza, Andreas Velten, and Mohit Gupta. Practical coding function design for time-of-flight imaging. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1566–1574, 2019. 16

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 17

[9] David B Lindell, Matthew O'Toole, and Gordon Wetzstein. Single-photon 3d imaging with deep sensor fusion. *ACM Trans. Graph.*, 37(4):113–1, 2018. 1, 13, 14, 15, 16

[10] Parsa Mirdehghan, Wenzheng Chen, and Kiriakos N Kutulakos. Optimal structured light à la carte. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6248–6257, 2018. 16

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019. 16

[12] Jiayong Peng, Zhiwei Xiong, Xin Huang, Zheng-Ping Li, Dong Liu, and Feihu Xu. Photon-efficient 3d imaging with a non-local neural network. In *European Conference on Computer Vision*, pages 225–241. Springer, 2020. 16

[13] Michael P. Sheehan, Julián Tachella, and Mike E. Davies. A sketching framework for reduced data transfer in photon counting lidar. *IEEE Transactions on Computational Imaging*, 7:989–1004, 2021. 11, 13, 14, 16

[14] Zhanghao Sun, David B Lindell, Olav Solgaard, and Gordon Wetzstein. Spadnet: deep rgb-spad sensor fusion assisted by monocular depth estimation. *Optics express*, 28(10):14948–14962, 2020. 16

[15] Arin Can Ulku, Claudio Bruschini, Ivan Michel Antolović, Yung Kuo, Rinat Ankri, Shimon Weiss, Xavier Michalet, and Edoardo Charbon. A 512× 512 spad image sensor with integrated gating for widefield flim. *IEEE Journal of Selected Topics in Quantum Electronics*, 25(1):1–12, 2018. 5

[16] Augusto Ronchini Ximenes, Preethi Padmanabhan, Myung-Jae Lee, Yuichiro Yamashita, Dun-Nian Yaung, and Edoardo Charbon. A modular, direct time-of-flight depth sensor in 45/65-nm 3-d-stacked cmos technology. *IEEE Journal of Solid-State Circuits*, 54(11):3203–3214, 2019. 4