Supplementary material for:
Hardware-Aware Photonic Fourier Processing for Automotive Radar

Niklas Bahr[1,2], Thomas Gisder[2], Fabian Otto[2], Jelle Dijkstra[1], Lennart Meyer[1], Falk Ebert[1], Linus Krämer[1], Daniel Steinmeyer[2], Frank Brückerhoff-Plückelmann[1], and Wolfram Pernice[1]

[1] *Kirchhoff-Institute for Physics, University of Heidelberg; Heidelberg, 69120, Germany.*
[2] *Group Innovation, Volkswagen AG; Wolfsburg, 38436, Germany*

Contents

Supplementary Text S3
S1. Fourier transforms S3
1. From discrete Fourier transform (DFT) to fast Fourier transform (FFT) S3
2. Higher-dimensional Fourier transforms S4
3. Stagewise FFT S4
S2. Weight reuse factors S5
1. Stagewise FFT S5
2. Dimensionwise FFT S7
3. Vector-radix FFT S7
S3. Analog-photonic matrix processing unit S9
1. Implementation of the stagewise FFT S9
2. Impact of stagewise FFT on computational time S9
3. Weight error S11
4. MVM error S11
5. One-dimensional testcases S12
S4. Frequency-modulated continuous wave radar S15
1. Experimental radar setting S16

Supplementary Text

S1. Fourier transforms

Considering a continuous time-series $x(t)$ its Fourier transform $\tilde{x}(\omega)$ is defined as:

$$\tilde{x}(\omega) = \int_{\mathbb{R}} x(t) \mathrm{e}^{-\mathrm{i}\omega \cdot t} \mathrm{d}t. \tag{1}$$

1. *From discrete Fourier transform (DFT) to fast Fourier transform (FFT)*

Calculating the Fourier transform numerically, one performs the discrete Fourier transform (DFT). Here, the signal $x(t)$ is discretized into a sequence of N equally distributed sample points $x = (x_0, \ldots, x_{N)-1}$. The DFT is a sequence of N discrete frequency domain sample points $\tilde{x} = (\tilde{x}_0, \ldots, \tilde{x}_{N-1})$. The components $\tilde{x}_k$ of the DFT are computed via

$$\tilde{x}_k = \sum_{n=0}^{N-1} x_n \mathrm{e}^{-\mathrm{i}\frac{2\pi}{N}k \cdot n} = \sum_{n=0}^{N-1} x_n W_N^{k \cdot n}, \tag{2}$$

where the coefficients W_N^j are called roots of unity, or in the context of Fourier transform the *twiddle factors*. The one-dimensional DFT can be expressed as a matrix–vector product, hence the computational costs to compute such a DFT scale quadratically with discretization, i.e. $\mathcal{O}(N^2)$.
However, the DFT matrix contains substantial redundancy because the twiddle factors are N-periodic. The fast Fourier transform (FFT) of Cooley–Tukey exploits this structure by recursively decomposing a transform of length N into two transforms of length $N/2$, corresponding to the even and odd indices of the input vector, until only 2-point DFTs remain. The resulting intermediate quantities E_k and O_k are then reused to compute the two output components $\tilde{x}_k$ and $\tilde{x}_{k+N/2}$ simultaneously through the butterfly operation

$$\begin{bmatrix} \tilde{x}_k \\ \tilde{x}_{k+N/2} \end{bmatrix} = \begin{bmatrix} 1 & W_N^k \\ 1 & -W_N^k \end{bmatrix} \cdot \begin{bmatrix} E_k \\ O_k \end{bmatrix}. \tag{3}$$

In the following, we denote the butterfly matrix in Eq. (3) by $\hat{W}_N^k$. To obtain the correct output ordering, the input indices are reordered by bit reversal, so that the butterfly-connected pairs k and $k + N/2$ become adjacent.
The recursive radix-2 decomposition introduces one layer of butterfly operations per recursion step. For an input of length N, this yields $\log_2 N$ stages. Each stage S_i applies butterflies only to disjoint pairs of components and can therefore be written as a blockdiagonal matrix of size $N \times N$ with butterfly matrices on its diagonal. The full DFT matrix is thus factorized as

$$F_1 = S_{\log_2 N} \cdot \ldots \cdot S_1, \tag{4}$$

which reduces the computational scaling from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. This construction is referred to as the radix-2 FFT. Figure S1, reproduced from the main manuscript, illustrates the radix-2 FFT scheme for $N = 8$, which requires

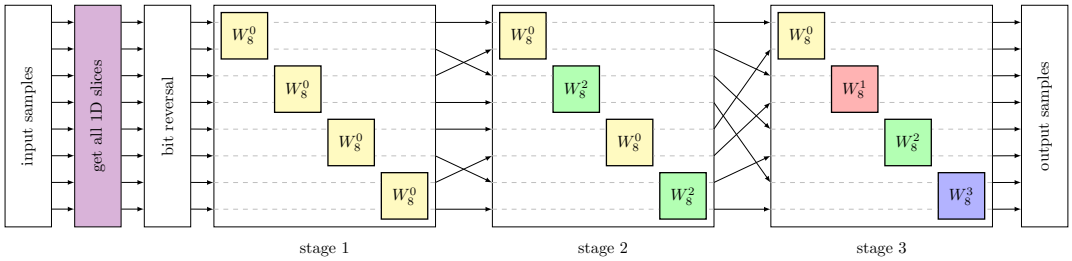

Figure S1. Scheme of the radix-2 fast Fourier algorithm on the example of an one-dimensional 8-point DFT. The FFT is computed in $\log_2 8 = 3$ stages, where each stage is represented by a block-diagonal matrix containing butterfly operations $\hat{W}_8^k$. The correct input ordering is obtained by bit-reversing the binary representation of the sample indices. In case of a higher-dimensional Fourier transform, we introduce a *stagewise* algorithm, where in advance all 1D slices of the data are computed, and then the algorithm is applied stage-by-stage on each slice simultaneously.

$\log_2 8 = 3$ stages. Here, one directly observes that some butterfly operators $\hat{W}_8^k$ are applied multiple times on different data points. Stage 1 contains only one butterfly operator corresponding to $\hat{W}_8^0$ that is repeated four times, stage 2 contains two butterfly operators corresponding to $\hat{W}_8^0,\ \hat{W}_8^2$ that are repeated twice each, and stage 3 contains four butterfly operators corresponding to $\hat{W}_8^0,\ \hat{W}_8^1,\ \hat{W}_8^2,\ \hat{W}_8^3$.

2. *Higher-dimensional Fourier transforms*

Typically Fourier transforms can also be applied to multidimensional scalar functions $x(t_0, ..., t_{d-1})$. Its Fourier transform is then defined via:

$$\tilde{x}(\omega_0, ...\omega_{d-1}) = \int_{\mathbb{R}^d} x(t_0, ..., t_{d-1}) \cdot \mathrm{e}^{-\mathrm{i}\sum_j \omega_j t_j} \cdot \mathrm{d}^d t \tag{5}$$

For the d-dimensional Fourier transform holds that it is separable into d 1-dimensional Fourier transforms along each dimension via the tensor product (Kronecker product):

$$F_d = F_{1,1} \otimes ... \otimes F_{1,d} \tag{6}$$

Here, we denote F_d as the d-dimensional Fourier transform and $F_{1,i}$ as the 1-dimensional Fourier transform along axis i.

3. *Stagewise FFT*

A higher-dimensional Fourier transform is the tensor product (Kronecker product) of one-dimensional Fourier transforms, see Eq. (6). Each one-dimensional Fourier transform can be decomposed via the FFT algorithm as a matrix product of the corresponding block-diagonal stage matrices, see Eq. (4). Due to properties of the Kronecker product, the execution order of tensor product and matrix product commutes:

$$(S_{1,1} \cdot\ S_{1,2}) \ \otimes\ (S_{2,1} \cdot\ S_{2,2}) = (S_{1,1}\ \otimes\ S_{2,1}) \cdot (S_{1,2}\ \otimes\ S_{2,2}). \tag{7}$$

This means that first stage 1 is being applied along all dimensions before applying stage 2 along all dimensions, and so on. This procedure we refer to as *stagewise* Fourier transform in contrast to the *dimensionwise* Fourier transform when sequentially calculating the complete fft along each dimension.

$$\bigotimes_{i=1}^{d} \left(\prod_{j=1}^{\log_2 N} S_{i,j} \right) = \prod_{j=1}^{\log_2 N} \left(\bigotimes_{i=1}^{d} S_{i,j} \right) \tag{8}$$

Both computation schemes, dimensionwise and stagewise, are referred to as variants of row-column radix-2 algorithms, in contrast to the vector-radix-2 algorithm that is being discussed in Sec. S2 3. The stagewise Fourier transform allows to maximize the reuse of butterflies and reduces the needs for reprogramming weights, which is extremely costly in time and energy on our photonic hardware.

S2. Weight reuse factors

In the row-column radix-2 algorithm, certain butterfly operators are applied repeatedly to different subsets of the input data, as illustrated in Fig. S1. This repetition is advantageous for hardware implementations: once a butterfly kernel is configured, it can be reused across all identical operations within a stage, thereby reducing both reconfiguration time and energy consumption. High weight reuses are necessary, especially for analog compute hardware, as weight programming dominates the overall energy requirements of such systems. This section derives the reuse factor R for the radix-2 case ($k = 2$) when implemented on a physical matrix processor of size $m \times m$, where $m := 2k = 4$. The relation $m = 2k$ arises because photonic hardware typically performs real-valued matrix–vector multiplications. Computing a complex $k \times k$ butterfly requires expanding real and imaginary components, resulting in a physical matrix of size $2k \times 2k$. Table S1, reproduced from the main manuscript, summarizes the reuse behavior of different FFT algorithms: dimensionwise radix-2, stagewise radix-2 and vector-radix-2.

	dimensionwise	stagewise	vector-radix
Butterfly size ($k = 2$)	2×2	2×2	$2^d \times 2^d$
MVMs	$\frac{d}{2} \cdot N^d \cdot \log_2 N$	$\frac{d}{2} \cdot N^d \cdot \log_2 N$	$(\frac{N}{2})^d \cdot \log_2 N$
Reprogrammings	$d \cdot N^{d-1} \cdot (N-1)$	$N-1$	$\frac{N^d-1}{2^d-1}$
Reuse factor R	$\frac{N}{2(N-1)} \cdot \log_2 N$	$\frac{d \cdot N^d}{2(N-1)} \cdot \log_2 N$	$\frac{N^d \cdot (2^d-1)}{2^d \cdot (N^d-1)} \cdot \log_2 N$

Table S1. Comparison of weight reuse between dimensionwise and stagewise implementation of Fourier transform. The total number of matrix–vector multiplications is identical in both cases. In the stagewise implementation, weights are reused across all 1D slices, causing the reuse factor to increase with both the dimension d and the number of sample points N.

1. *Stagewise FFT*

For d-dimensional Fourier transforms, weight reuse R is maximized when the transform is executed in a *stagewise* manner, ensuring that each kernel remains active for as long as possible before switching to the next stage. In the stagewise schedule, the outer loop iterates over the block-diagonal stage matrices, while the inner loop iterates over the one-dimensional slices of the underlying input tensor, see right-hand-side of Eq. (8). The number of such 1D slices in a d-dimensional tensor with N data points along each axis is

$$\#\text{slices} = N^{d-1} \cdot d. \tag{9}$$

Figure S2 illustrates this for a three-dimensional tensor with $N = 3$ and $d = 3$. Each axis contains $3^2 = 9$ one-dimensional slices, resulting in a total of $3 \cdot 9 = 27$ slices.

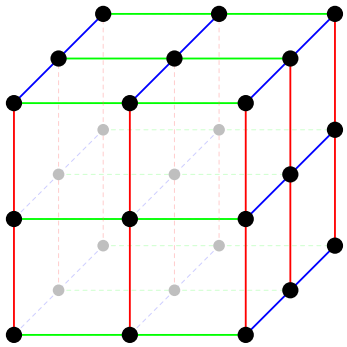

Figure S2. Illustration of 3-dimensional tensor of size $3 \times 3 \times 3$. Each data point is indicated via a black dot. One must count all 1D slices along each axis. Here: x-axis (green), y-axis (blue), and z-axis (red). In this example, there are 27 1D slices.

The radix-2 FFT consists of

$$\#\text{stages} = \log_2 N, \tag{10}$$

stages, and each stage contains

$$\#\text{butterflies per stage} = \frac{N}{2} \tag{11}$$

butterflies per stage. The total number of butterfly operations is therefore

$$\#\text{butterflies} = \#\text{stages} \cdot \#\text{butterflies per stage} = \frac{N \cdot \log_2 N}{2}. \tag{12}$$

The number of required weight reconfigurations is significantly smaller than the total number of butterflies. What matters is how many *different* butterfly kernels appear in each stage. As illustrated in Fig. S1, the radix-2 FFT exhibits a characteristic pattern:

- In stage 1, the butterfly $\hat{W}_8^0$ appears four times ($N/2$)
- In stage 2, two distinct butterflies ($\hat{W}_8^0$, $\hat{W}_8^2$) appear twice ($N/4$)
- In stage 3, four distinct butterflies ($\hat{W}_8^0$, $\hat{W}_8^1$, $\hat{W}_8^2$, $\hat{W}_8^3$) appear once each ($N/8$).

In general, stage s contains 2^{s-1} different butterflies, each used $N/2^s$ times. Consequently, the number of weight configurations is the number of different butterflies accumulated for each stage:

$$\#\text{reprogrammings} = \sum_{s=1}^{\log_2 N} 2^{s-1} = N - 1. \tag{13}$$

The average reuse factor for the stagewise FFT follows from the number of 1D slices, the number of butterflies, and the number of required reprogrammings:

$$R_{\text{stagewise}} = \frac{\#\text{slices} \cdot \#\text{butterflies}}{\#\text{reprogrammings}} = \frac{N^d \cdot d \cdot \log_2 N}{2 \cdot (N-1)}. \tag{14}$$

Stagewise FFT (scaling)

The radix-2 algorithm can be generalized to radix-k algorithms where k must be a power of two. Larger radices become relevant, when the programmable analog hardware provides larger physical matrix sizes m. With a larger matrix size, butterfly operations exceeding the 2×2 radix-2 case can be computed. In our case, the photonic processor implements real-valued matrix–vector multiplications, and thus complex butterflies of size $k \times k$ still require a matrix of size $m = 2k$.
Increasing the radix reduces the number of FFT stages. For a radix-k decomposition, the number of stages is

$$\#\text{stages} = \log_k N. \tag{15}$$

Each stage contains

$$\#\text{butterflies per stage} = \frac{N}{k} \tag{16}$$

butterfly operations. As in the radix-2 case, many butterflies within a stage are identical and can be reused. The number of distinct butterflies per stage is k^{s-1}, and summing over all stages yields the total number of required weight programmings:

$$\#\text{reprogrammings} = \sum_{s=1}^{\log_k N} k^{s-1} = \frac{N-1}{k-1}. \tag{17}$$

For higher-dimensional Fourier transforms, the number of 1D slices remains

$$\#\text{slices} = N^{d-1} \cdot d. \tag{18}$$

Combining these expressions gives the average weight-reuse factor for a radix-k FFT executed in a stagewise manner:

$$R_{\text{upscaled}} = \frac{\#\text{slices} \cdot \#\text{butterflies}}{\#\text{reprogrammings}} = \frac{\log_k N \cdot N^d \cdot d \cdot (k-1)}{k \cdot (N-1)}. \tag{19}$$

The physical matrix size and the problem size are constrained by $m = 2k$, with $4 \leq m \leq 2N$.

A consistency check confirms that Eq. (19) reproduces the expected limiting cases. For $k = N$, corresponding to a single-stage DFT, the reuse factor reduces to $N^{d-1} \cdot d$, which is the number of 1D slices, see also Eq. (9). For $k = 2$, Eq. (19) reduces to the radix-2 expression in Eq. (14). For intermediate values of k, the reuse factor lies between these two extremes. The lowest reuse occurs for the DFT case $k = N$, which we identify as the worst case from a reuse perspective: it requires only a single time step and therefore has minimal latency, but also minimal reuse, see also Fig. 2 (c) in the main manuscript. In contrast, the radix-2 case yields the highest reuse but requires the largest number of matrix–vector multiplications.

Stagewise FFT (tiling)

Matrix tiling is necessary when the available hardware can not process a full butterfly matrix in a single operation. This limitation is common for large matrices, such as those in fully connected neural-network layers. For implementing the radix-2 algorithm on photonic hardware, an analogous limitation appears: the required butterfly matrix has a size of 4×4, whereas the hardware used in this work realizes a 4×1 configuration. The butterfly must therefore be decomposed into four tiles corresponding to its rows, which are computed sequentially and recombined to form the complete output vector.
This tiling procedure increases the number of required weight reconfigurations by a factor of $t = 4$, and reduces the average weight reuse factor. The reduction arises because all butterfly operations along one dimension must be completed before the tiles of the next dimension can be processed. For large sample sizes N, and d-dimensional FFTs, however, the average weight reuse remains substantial, since it scales with the number of 1D slices N^{d-1}.
For the dimensionwise schedule, tiling does not affect the reuse factor. In that case, weight are reconfigured after each stage regardless the number of 1D slices.

	dimensionwise	stagewise
total MVMs	$d \cdot N^{d-1} \cdot \log 2(N) \cdot \frac{N}{2} \cdot \mathbf{t}$	$d \cdot N^{d-1} \cdot \log 2(N) \cdot \frac{N}{2} \cdot t$
reprogrammings	$d \cdot N^{d-1} \cdot (N-1) \cdot \mathbf{t}$	$(N-1) \cdot t \cdot d$
reuse factor	$\frac{\log 2(N) N}{2(N-1)}$	$\frac{\log 2(N) \cdot N^d}{2(N-1)}$

Table S2. Comparison of weight reuse between dimensionwise and stagewise implementation of Fourier transform in case of tiling. The butterfly matrix is decomposed in t tiles. With this, the factor d in the stagewise implementation is lost, while there are no changes in the dimensionwise implementation.

2. Dimensionwise FFT

In the *dimensionwise* case, the weights are reconfigured each stage and does not reuse them across the different 1D slices. Its reuse factor is, therefore,

$$R_{\text{dimensionwise}} = \frac{\#\text{butterflies}}{\#\text{reprogrammings}} = \frac{N \cdot \log_2 N}{2 \cdot (N-1)}. \tag{20}$$

Equation (20) is identical to the stagewise expression for $d = 1$, consistent with Eq. (14). This highlights that the stagewise FFT exploits its high reuse factor from the usage of identical operations on multiple 1D slices in higher-dimensional Fourier transforms.

3. Vector-radix FFT

An alternative of the radix-2 algorithm for multidimensional Fourier transforms is the vector-radix-k algorithm. While the radix-k approach evaluates a discrete Fourier transform by applying one-dimensional FFTs along each axis, the vector-radix-k formulation treats the transform directly in d dimensions.
For an input array of size $N \times \cdots \times N$, the vector-radix-2 decomposition splits the original d-dimensional FFT into 2^d smaller d-dimensional FFTs of size $\frac{N}{2} \times \cdots \times \frac{N}{2}$. This is the multidimensional analogue of row-column radix-2 decimation. In two dimensions, the decomposition yields four subtransforms associated with the parity classes (E, E), (O, E), (E, O), and (O, O). In d dimensions, the same construction gives 2^d parity combinations and hence 2^d subtransforms. In both cases, vector-radix and row-column, the algorithm takes $\log_2 N$ global stages. However, the vector-radix algorithm requires a physical kernel of complex size $2^d \times 2^d$, whereas the standard radix-2 algorithm only requires a kernel size of 2×2.

For vector-radix-2, a fixed stage contains N^d datapoints. One vector-radix butterfly combines 2^d points at once. The total number of butterfly operations is

$$\#\text{butterflies} = \frac{N^d}{2^d} \log_2 N. \tag{21}$$

While in the one-dimensional row-column scheme, the butterfly operators within a given stage are distinguished by a single twiddle factor W_N^j, and 2^{s-1} distinct butterfly configurations on stage s, see Eq. (13), the implementation of a d dimensional vector-radix requires butterfly operators with one such index along each axis, so that each stage is characterized by combinations of indices $j_1, \ldots, j_d$. Compared with dimensionwise radix-2 FFTs, the number of settings is much larger because the twiddle factors contain mixed terms such as $W_N^{j_1+j_2}$ in two dimensions
The number of different butterfly settings thus grows more rapidly with dimensionality for the vector-radix algorithm. The total number of different twiddle configurations at that stage is therefore given by $2^{d(s-1)}$. For vector-radix-2, the total number of required reprogrammings is

$$\#\text{reprogrammings} = \sum_{s=1}^{\log_2 N} 2^{d(s-1)} = \frac{N^d - 1}{2^d - 1}. \tag{22}$$

The average reuse factor is therefore

$$\#\text{reuse} = \frac{\#\text{butterflies}}{\#\text{reprogrammings}} = \frac{N^d (2^d - 1) \log_2 N}{2^d (N^d - 1)}. \tag{23}$$

As shown in the main manuscript, the reuse factor of the vector-radix scheme is higher than the dimensionwise scheme, but significantly lower than the stagewise scheme.

S3. Analog-photonic matrix processing unit

1. Implementation of the stagewise FFT

Our analog-photonic processor performs real-valued matrix–vector multiplications of size 4×1. For implementing the radix-2 stagewise scheme, the complex 2×2 butterfly is therefore expanded into an equivalent real 4×4 matrix. By separating real and imaginary components, the butterfly matrix

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & W_N^k \\ 1 & -W_N^k \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{24}$$

is rewritten as the real-valued 4×4 representation

$$\begin{bmatrix} \mathrm{Re}(y_1) \\ \mathrm{Im}(y_1) \\ \mathrm{Re}(y_2) \\ \mathrm{Im}(y_2) \end{bmatrix} = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 0 & +\mathrm{Re}(W_N^k) & -\mathrm{Im}(W_N^k) \\ 0 & 1 & +\mathrm{Im}(W_N^k) & +\mathrm{Re}(W_N^k) \\ 1 & 0 & -\mathrm{Re}(W_N^k) & +\mathrm{Im}(W_N^k) \\ 0 & 1 & -\mathrm{Im}(W_N^k) & -\mathrm{Re}(W_N^k) \end{bmatrix} \cdot \begin{bmatrix} \mathrm{Re}(x_1) \\ \mathrm{Im}(x_1) \\ \mathrm{Re}(x_2) \\ \mathrm{Im}(x_2) \end{bmatrix}. \tag{25}$$

With the hardware used in this work, the matrix $\hat{W}_N^k$ cannot be applied in a single hardware step and therefore must be executed in four sequential tiles. The implementation processes these four rows one after another and recombines the intermediate results digitally. For any fixed axis of the data tensor, each tensor element participates in exactly one butterfly operation per stage, which allows the stage to be decomposed into independent tiles without violating data dependencies. To maximize reuse of each weight tile, a local copy of the input tensor is created, and the tile is applied to all 1D slices along the chosen dimension and to all relevant butterfly positions within each slice. The intermediate results are written into the local copy, and after all tiles have been processed, the fully updated local copy replaces the original input. The procedure is then repeated for the next dimension. As discussed in Sec. S2 1, this tiling strategy reduces weight reuse by a factor of d compared to the originally proposed stagewise radix-2 algorithm. Algorithm 1 shows pseudo code demonstrating the implemented stagewise tiling algorithm.

Algorithm 1 Stagewise FFT (tiling algorithm)

1: **Input:** d-dimensional complex-valued tensor with 1D slices of length N
2: Extract 1D slices; apply bit-reversal to each slice
3: **for** $s = 1$ to $\log_2 N$ **do**
4: $m \leftarrow 2^s$
5: half $\leftarrow m/2$
6: **for** $k = 0$ to half-1 **do**
7: $k_{\mathrm{twiddle}} \leftarrow k \cdot N/m$
8: Construct butterfly matrix $\hat{W}_N^{k_{\mathrm{twiddle}}}$
9: **for** dim $= 1$ to d **do**
10: **for** tile $= 1$ to t **do**
11: Load tile $\hat{W}_N^{k_{\mathrm{twiddle}}}$[tile]
12: **for** each 1D slice along dimension dim **do**
13: Expand slice to local array x of length $2N$
14: **for** $j = 0$ to $N/m - 1$ **do**
15: $p \leftarrow j \cdot m + k$
16: $q \leftarrow p +$ half
17: Apply tile to $(x_{2p}, x_{2p+1}, x_{2q}, x_{2q+1})$
18: Store result in local copy of x
19: Write local copy back to input data

If larger hardware were available, no tiling is required anymore. In this case, the algorithm simplifies as the butterfly matrix can be directly loaded into the hardware in line 8 and then be reused for all dimensions and slices. Second, the butterfly results then can be written in-place into the input tensor without creating local copies of the tensor.

2. Impact of stagewise FFT on computational time

As the dimensionwise and stagewise implementation of the FFT represent the exact idetical arithmetic operation, a result-based comparision of both algorithms will not yield any differences. Here, we investigate the compute

time of both implementations to proof the advantage of stagewise FFTs on analog-photonic hardware. On our specific hardware the benefit of weight reuse becomes strongly visible and practically relevant, as weight programming dominates the system's behavior ($t_{\text{weight}} \gg t_{\text{MVM}}$).

The dimensionwise and stagewise FFT schedules implement the same arithmetic transform and are expected to produce the same result, up to analog noise and finite hardware precision. The relevant distinction is, instead, the execution cost on reconfigurable analog-photonic hardware. We therefore evaluate the computation time of both schedules to validate the hardware-efficiency claim of the stagewise FFT. In the present processor, this effect is particularly pronounced because programming a new weight configuration is much slower than executing one matrix–vector multiplication ($t_{\text{weight}} \gg t_{\text{MVM}}$).

The total FFT execution time on the analog-photonic processor is modelled as the sum of configuration time and MVM execution time,

$$\tau_{\text{FFT}} = \#\text{reprogrammings} \cdot \tau_{\text{weight}} + \#\text{MVMs} \cdot \tau_{\text{MVM}}, \tag{26}$$

where the counts of reprogrammings and MVMs are given in Tab. S2 for stagewise and dimensionwise FFT in the case of tiling. The hardware-characteristic times τ_{weight}, τ_{MVM} for reconfiguring the weight kernel and performing a single MVM, respectively, are obtained from experimental data and fitting. With N denoting the input sample size, d the transform dimensionality, and t the execution time of the stagewise FFT is

$$\tau_{\text{stagewise}}(N, d, t) = d \cdot t \cdot (N-1) \cdot \tau_{\text{weight}} + d \cdot N^{d-1} \cdot t \cdot \log_2 N \cdot N/2 \cdot \tau_{\text{MVM}}. \tag{27}$$

For dimensionwise execution, the MVM count is identical, but the number of physical weight reconfigurations increases by a factor of N^{d-1},

$$\tau_{\text{dimensionwise}}(N, d, t) = d \cdot t \cdot (N-1) \cdot N^{d-1} \cdot \tau_{\text{weight}} + d \cdot N^{d-1} \cdot t \cdot \log_2 N \cdot N/2 \cdot \tau_{\text{MVM}}. \tag{28}$$

For the two-dimensional case evaluated here, with $d = 2$, $t = 4$, and $N \leq 128$, the experimentally fitted hardware parameters are $\tau_{\text{weight}} \approx 128$ ms, and $\tau_{\text{MVM}} \approx 0.72$ ms. The fitted parameters are nearly identical for stagewise and dimensionwise execution, confirming that the two schedules differ primarily in the number of weight reconfiguration events. Figure S3 compares the measured execution times with the model for two-dimensional radix-2 FFTs implemented on the analog-photonic processor using 4×1 hardware tiles and averaging over 2^4 repeated measurements per MVM. The measured scaling follows the predicted dependence on the number of reconfigurations. This confirms that for stagewise FFT reusing programmed FFT-stage weights substantially reduces the weight-programming overhead on reconfigurable analog-photonic hardware. Furthermore, we observe that without the stagewise ordering, the configuration-time overhead would make the larger two-dimensional FFTs impractical on the present experimental platform.

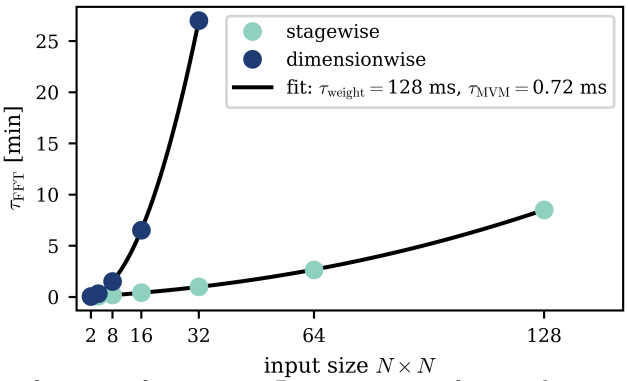

Figure S3. FFT processing time as a function of input size. Processing time for two-dimensional radix-2 FFTs of size $N \times N$ on the analog-photonic processor. Each 4×1 MVM operation is averaged over 2^4 repeated measurements. The measured runtimes for stagewise and dimensionwise scheduling follow the models in Eq. (27) and Eq. (28), respectively, confirming that stagewise execution reduces runtime by reducing the number of physical weight reconfigurations at identical MVM count.

3. *Weight error*

The weight-setting error of the analog-photonic hardware is estimated via:

$$\Delta_{\text{weight}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} |w_{\text{i,exact}} - w_{\text{i,measured}}|^2} \tag{29}$$

This is translated into the peak signal-to-noise ratio (PSNR)

$$\text{PSNR} = -20 \cdot \log_{10}(\Delta_{\text{weight}}) \tag{30}$$

From this we estimate an effective number of bits for the weight resolution:

$$\text{ENOB} = \frac{\text{PSNR} - 1.76}{6.02} \tag{31}$$

From 4.88 bit we expect to distinguish approximately $2^{4.88} \approx 29$ different twiddle weights. However, the system can resolve approximately 4 times these twiddle factors because of the four quadrants with different signs in real and imaginary part, see also Fig. S5.

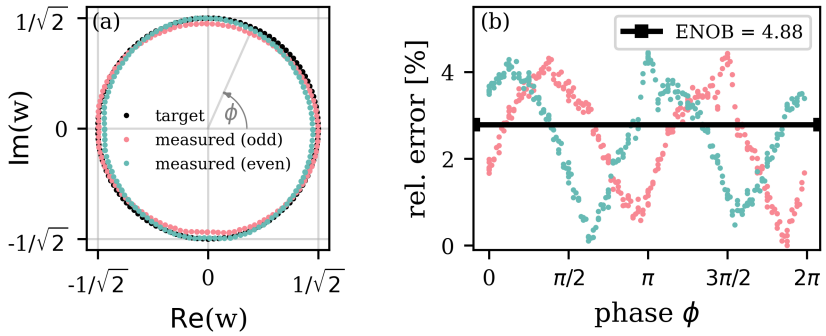

Figure S4. Weight-setting error on analog-photonic hardware. (a) The complex-valued twiddle weights of the normalized radix-2 butterfly lie on a circle with radius $1/\sqrt{2}$. (b) Complex twiddle weights are mapped onto two real-valued hardware channels, so the weight-setting error depends on whether the real or imaginary component is assigned to the even or odd channel. The error oscillates with the twiddle angle and corresponds, on average, to an effective number of bits of 4.88 bit.

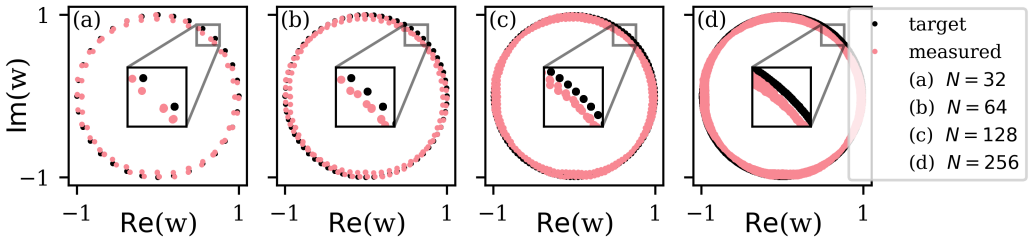

Figure S5. Twiddle-weight distribution for different transform sizes N from 32 to 256. As N increases, the twiddle factors sample the unit circle more densely, reducing the separation between adjacent weights. This limits the effective resolution available for analog-photonic Fourier processing.

We further estimate that the full FFT-error will increase with number of sample points, as the twiddle weights can not fully be distinguished, and error propagates through multiple stages of the FFT.

4. *MVM error*

Because the matrix–vector multiplications are performed in the analog domain, each MVM is affected by finite precision and noise. Relevant error sources include optical noise from the amplified-spontaneous-emission source

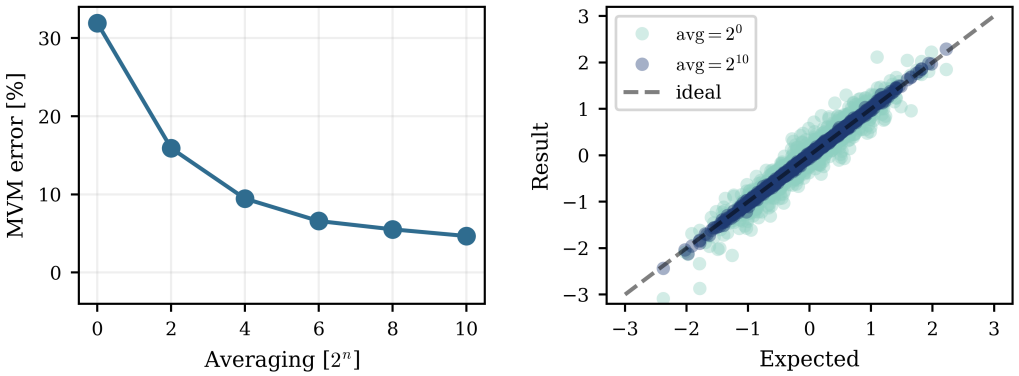

Figure S6. Effect of averaging on analog-photonic MVM accuracy. Relative MVM error σ_{MVM} of 4×1 operations as a function of the number of averaged measurements (left). Scatter plot of target and measured MVM outputs for different averaging settings, evaluated over 1000 randomly generated input and weight configurations (right).

(Agilent 83438A), electronic readout noise, finite weight-setting accuracy and other hardware nonidealities. To reduce the stochastic component of the error, each 4×1 MVM operation is repeated multiple times and the measured outputs are averaged before further processing. This averaging reduces noise-induced deviations, whereas systematic contributions such as residual weight errors are not removed by repeated readout. Figure S6 (a) shows the resulting MVM error σ_{MVM} as a function of the number of averaged measurements. Figure S6 (b) compares target and measured MVM outputs for 1000 randomly generated input and weight configurations, using the minimum and maximum averaging settings. The reduced scatter at higher averaging illustrates the suppression of stochastic readout noise. The MVM error σ_{MVM} is quantified as the mean relative L_2 deviation from the digital reference

$$\sigma_{\mathrm{MVM}} = \frac{\langle \| x_{\mathrm{target},i} - x_{\mathrm{measured},i} \|_2 \rangle_i}{\langle \| x_{\mathrm{target},i} \|_2 \rangle_i}. \tag{32}$$

5. *One-dimensional testcases*

To validate the implementation of Algorithm 1 on our photonic hardware, we used a linearly chirped cosine signal with amplitude

$$A(t) = A_0 \cdot \cos\left(2\pi f_0 t + \pi \beta t^2\right). \tag{33}$$

Here, the frequency increases linearly as $f(t) = f_0 + \beta \cdot t$, with chirp rate $\beta = \frac{f_1 - f_0}{T}$ for start frequency f_0, end frequency f_1, and chirp duration T.
We chose a chirp spanning $f_0 = 1$ (a.u) to $f_1 = 8$ (a.u), with chirp duration of $T = \frac{N}{32}$. The reference case with $N = 32$ sample points therefore corresponds to a time interval 0 to 1. For other sample numbers N, the adjusted chirp duration preserves the temporal bin size while increasing the spectral resolution proportionally to N, see Fig. S7.
We computed the FFT of this signal for different sample numbers N and compared the corresponding photonic results with the exact digital reference. For this purpose, we evaluated the normalized magnitude spectra $|\tilde{A}|$ and defined the root-mean-square magnitude error as

$$\Delta_{\mathrm{fft}} = \sqrt{\frac{1}{N} \sum_{k=1}^{N} (|\tilde{A}_{k,\mathrm{opt}}| - |\tilde{A}_{k,\mathrm{exact}}|)^2}. \tag{34}$$

In radar applications, peaks in the magnitude spectrum indicate likely target locations. Furthermore, we validated the implementation using Parseval's theorem, which expresses energy conservation for a unitary transform. Unitarity is achieved by introducing the scaling factor $1/\sqrt{2}$ in Eq. (25).With this normalization, the total signal energy is conserved over all stages and Parseval's theorem takes the form

$$\sum_{k=1}^{N} |A_k|^2 = \sum_{k=1}^{N} |\tilde{A}_k|^2. \tag{35}$$

Detailed results are shown in Fig. S8.

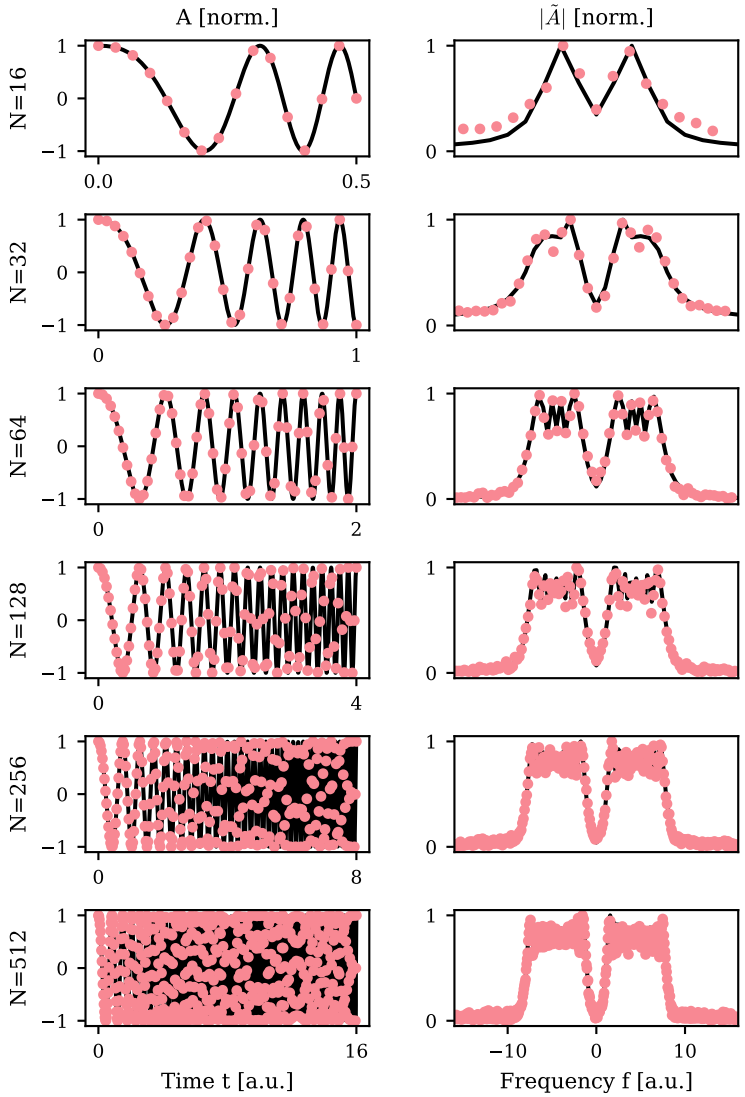

Figure S7. Time- and frequency-domain representations of the chirped signals considered in this study. Orange dots indicate the sampled points used for photonic computation of the Fourier transform (left) and the corresponding spectral samples (right). The $N = 64$ case is shown in the main manuscript.

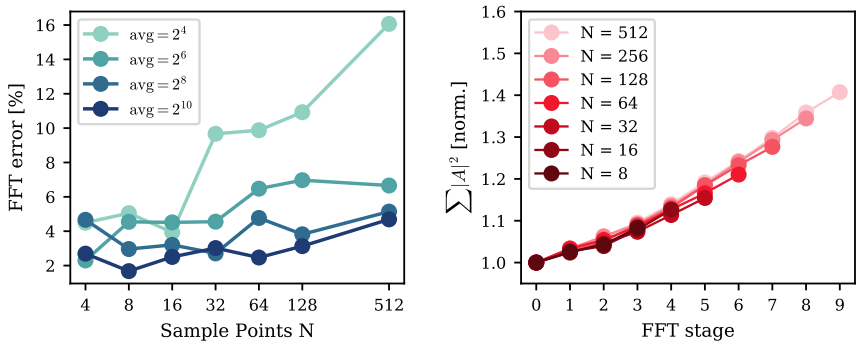

Figure S8. FFT error and Parseval analysis for different averaging levels and sample numbers. As expected, the FFT error increases with the number of sample points, but can be reduced by repeated measurements that average out light-source-induced noise (left). The signal energy increases slightly with the number of FFT stages, indicating a small normalization bias in the hardware. After normalization, the correct signal shape is preserved (right).

Overall, the chirped spectrum is reproduced in good agreement with the digital reference. As expected, the FFT error increases with the number of sample points N, owing to limited weight precision and error accumulation over the increasing number of stages $\log_2 N$. This trend is also reflected in the Parseval analysis. In principle, the energy is conserved, but it increases slightly with the number of stages, indicating tendency of the system to overestimate the signal amplitude. This effect is compensated in post-processing by normalizing the resulting feature map. With sufficiently high averaging, the FFT error is reduced substantially. This limitation is therefore not algorithmic, but arises from the noisy light source and other imperfections used in the present hardware setup.

S4. Frequency-modulated continuous wave radar

Frequency-modulated continuous-wave (FMCW) radar systems transmit chirps with linearly increasing frequency:

$$f(t) = f_0 + \frac{B}{T_c} \cdot t. \tag{36}$$

The pulse travels through space, gets reflected on a target and reaches the receiver after time

$$\tau_0 = \frac{2 \cdot d}{c_0}. \tag{37}$$

We assume the far-field regime, $d \gg \lambda$. For a radar frequency of $f = 77\,\mathrm{GHz}$, corresponding to $\lambda \approx 4\,\mathrm{mm}$, targets at much larger distances can be treated as producing approximately planar wavefronts. The received time-delayed signal is mixed with the currently transmitted signal.

$$T_x(t) \cdot R_x^*(t - \tau_0) = e^{\mathrm{i}2\pi f \cdot t} \cdot e^{-\mathrm{i}2\pi f \cdot (t-\tau_0)} = e^{\mathrm{i}2\pi f \cdot \tau_0} = e^{\mathrm{i}(2\pi(f_0 + \frac{B}{T_c}t)\cdot\tau_0)} = e^{\mathrm{i}2\pi f_0 \cdot \tau_0} \cdot e^{\mathrm{i}2\pi \frac{B\cdot\tau_0}{T_c}\cdot t} \tag{38}$$

The schematic signal flow for such an FMCW radar system is shown in Fig. S9.

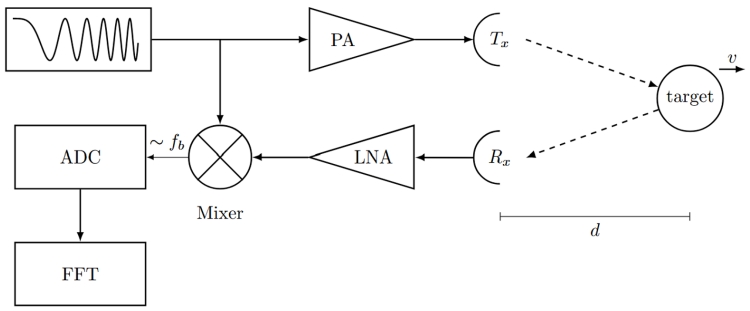

Figure S9. Signal flow of a FMCW radar system.

We define the beat frequency f_b, which gives a direct link to the distance d of the object as:

$$f_b \coloneqq \frac{B \cdot \tau_0}{T_c} = \frac{2 \cdot B \cdot d}{c_0 \cdot T_c}. \tag{39}$$

The global phase of the received signal is given via

$$\phi_0 \coloneqq 2\pi \cdot f_0 \cdot \tau_0 = \frac{4\pi \cdot f_0 \cdot d}{c_0}. \tag{40}$$

By sending multiple pulses, for a moving object the distance varies by $\Delta d = v \cdot T_c$ from pulse to pulse and hence the global phase varies by

$$\Delta\phi = \frac{4\pi f_0 v T_c}{c_0} \tag{41}$$

This phase changes with time, resulting in another oscillation with Doppler-frequency:

$$f_d = \frac{1}{2\pi} \frac{\Delta\phi}{\Delta t|_{T_c}} = \frac{2 f_0 v}{c_0} \tag{42}$$

The temporal delay between transmitted and received chirp and the corresponding beat frequency f_b and the global phase difference $\Delta\phi$ of two received chirps are illustrated in Fig. S10. In an FMCW radar, the received signal is sampled along two time axes. The *fast-time* axis corresponds to the samples acquired within a single chirp and therefore contains the beat-frequency f_b information used for range estimation. The *slow-time* axis corresponds to the sequence of repeated chirps and captures the phase evolution from chirp to chirp, enabling velocity estimation through the Doppler frequency f_d. A 2D FFT across these axes yields the range and velocity response of the scene. With multiple transmit and receive antennas, lateral and vertical angle information can be resolved as well, extending the processing to a 4D FFT

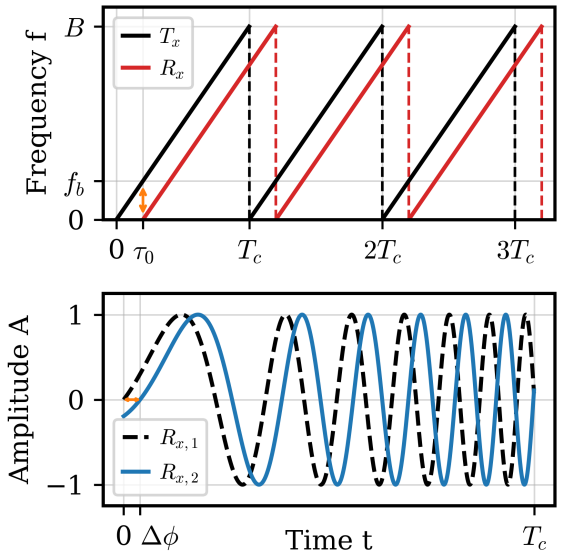

Figure S10. Chirped signals transmitted and received from the radar system.

1. *Experimental radar setting*

In our experimental setup, we used a 77 GHz FMCW radar sensor of the company INRAS (*radarlog*) to transmit and receive radar signals. Furthermore, we used a 76-81 GHz Transceiver (National Instruments mmRH-3608) to emulate moving objects. This device detects the emitted radar wave from the *radarlog* and delays it to simulate a moving object at a certain range and velocity. The values in Tab. S3 describe the raw radar acquisition before downsampling. For analog-photonic processing, the acquired data were downsampled to 128 × 128.

Parameter	Symbol	Value	Formula
Carrier frequency	f_0	76.5 GHz	
Start frequency	f_{start}	76 GHz	
Stop frequency	f_{stop}	77 GHz	
Bandwidth	B	1 GHz	
Up-Chirp duration	$T_{\text{c,up}}$	51.2 μs	
Down-Chirp duration (dead time)	$T_{\text{c,down}}$	8.8 μs	
Chirp repetition interval	T_{c}	60 μs	$T_{\text{c}} = T_{\text{c,up}} + T_{\text{c,down}}$
Number of samples per chirp	N	256	
Number of chirps	N_c	256	
(Vacuum) speed of light	c	$3 \cdot 10^8$ m/s	
Sampling rate (fast-time)	f_{fast}	5 MHz	$\frac{N}{T_{\text{c,up}}}$
Sampling rate (slow-time)	f_{slow}	16.67 kHz	$\frac{1}{T_{\text{c}}}$
Up-chirp slope	S	19.53 MHz/μs	$\frac{B}{T_{\text{c,up}}}$
Beat-frequency resolution	Δf_{b}	19.53 kHz	$\frac{f_{\text{fast}}}{N}$
Doppler-frequency resolution	Δf_{d}	65.1 Hz	$\frac{f_{\text{slow}}}{N_{\text{c}}}$
Range resolution	Δd	0.15 m	$\frac{c}{2B}$
Maximum unambiguous range	$d_{\max}$	19.2 m	$\frac{cf_{\text{fast}}}{4S}$
Velocity resolution	Δv	0.128 m/s	$\frac{c}{2f_0 N_{\text{c}} T_{\text{c}}}$
Maximum unambiguous velocity	$v_{\max}$	16.3 m/s	$\frac{c}{4f_0 T_{\text{c}}}$

Table S3. Radar waveform and acquisition parameters used in the FMCW experiment.

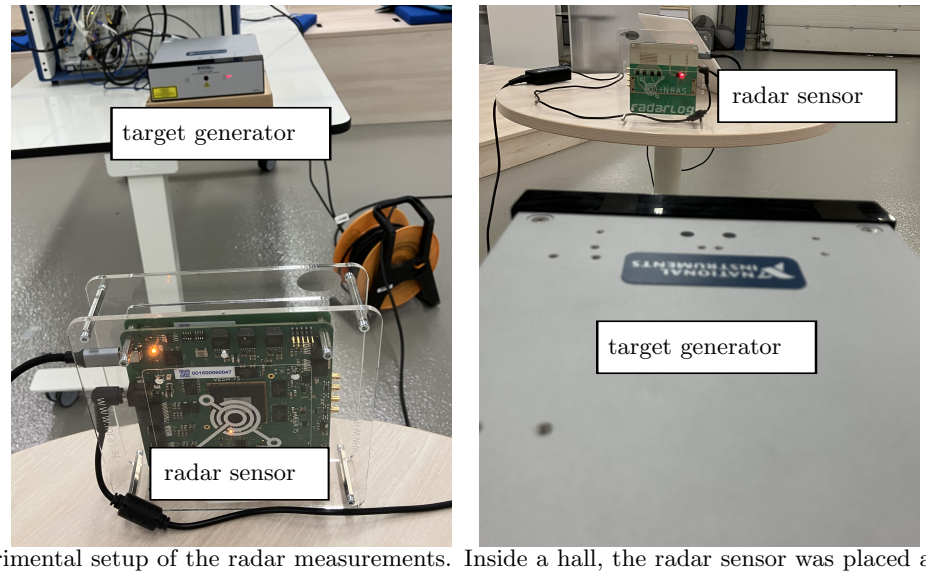

Figure S11. Experimental setup of the radar measurements. Inside a hall, the radar sensor was placed approximately 1.5 m in front of a target generator, which detects the radar wave and simulates a moving object.