Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Hardware-Aware Photonic Fourier Processing for Automotive Radar

Wolfram Pernice

wolfram.pernice@kip.uni-heidelberg.de

Heidelberg University https://orcid.org/0000-0003-4569-4213

Niklas Bahr

Heidelberg University

Thomas Gisder

Volkswagen AG

Fabian Otto

Volkswagen AG

Jelle Dijkstra

Heidelberg University https://orcid.org/0009-0007-4647-4235

Lennart Meyer

Heidelberg University https://orcid.org/0009-0000-9381-9856

Falk Ebert

Ruprecht-Karls-Univ Heidelberg

Linus Krämer

Heidelberg University

Daniel Steinmeyer

Volkswagen AG https://orcid.org/0000-0001-5801-6951

Frank Brückerhoff-Plückelmann

University of Heidelberg https://orcid.org/0009-0009-7713-6735

Article

Keywords:

Posted Date: November 23rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11706200/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: There is **NO** Competing Interest.

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.*
Email: [†]*wolfram.pernice@kip.uni-heidelberg.de*

Fourier transforms are central to real-time sensing, including imaging, wireless communication, and radar. In automotive systems, growing resolution, sensor count, and frame rate make Fourier processing demanding under strict energy and latency constraints. Analog-photonic processors are attractive because they execute linear operations with high efficiency, high bandwidth, and low latency. However, efficient Fourier processing on analog hardware is limited by analog–digital conversion and repeated weight programming. Here, we introduce a stagewise algorithm for hardware-aware multidimensional Fourier processing on programmable analog hardware. By reordering computation stage-by-stage across dimensions, it reuses each programmed kernel across many operations, increasing weight reuse by orders of magnitude and reducing reprogramming cost compared with conventional schedules. We implement this approach on an analog-photonic processor and demonstrate range–velocity processing of automotive radar data as a practically relevant use case. Our results show that algorithmic reordering makes multidimensional Fourier workloads system-efficient on programmable analog hardware.

Introduction

Fourier transforms are a fundamental operation for analyzing structured signals in the frequency domain. They form the basis of spectral analysis, filtering, and convolution, and have applications in image processing, wireless communication, and radar[1,2]. The fast Fourier transform (FFT) makes such computations efficient by reducing the arithmetic complexity of the discrete Fourier transform from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$[3,4]. In real-time sensing systems, however, the challenge is not only arithmetic complexity: Fourier transforms must be evaluated continuously on streaming data under strict energy, memory-bandwidth, and latency constraints. Automotive radar provides a representative example in which range and relative-velocity information is used for environment perception and driver-assistance systems, for example to identify a preceding vehicle that is slowing down, as illustrated in Fig. 1a. In frequency-modulated continuous-wave (FMCW) and multiple-input multiple-output (MIMO) radar, sampled beat signals are organized into data tensors, and multidimensional FFTs extract range, velocity, and angular information[5–7]. As radar systems move towards higher resolution, higher frame rate, and larger antenna arrays, the associated Fourier processing workload becomes an increasing system-level bottleneck. Conventional responses, such as scaling local compute in the vehicle or using cloud-computing resources, can alleviate this bottleneck only within the latency constraints of safety-critical perception and the limited energy budget of automated vehicles[8].

FFT implementations span different hardware platforms and application requirements. Highly optimized libraries deliver strong performance on general-purpose processors[9–11], while application-specific integrated circuits provide higher efficiency for fixed workloads[12]. FFT accelerators have been demonstrated in digital CMOS[13,14], analog CMOS[15,16], in-memory[17,18], and analog-optical platforms[19–21]. Analog and optical FFT implementations include charge-domain switched-capacitor networks[15], current-mode current-mirror circuits[16], and optical diffractive structures[20]. However, in such analog and optical FFT systems, the Fourier transform is largely encoded in the circuit topology, and this has so far been demonstrated mainly on a small scale, with transform sizes up to $N = 16$, limiting their flexibility as programmable processors. By contrast, programmable analog[22–24] and analog-photonic processors[25–27] are attractive in this context because they perform energy-efficient[28–31], high-bandwidth linear operations with low latency[32,33] and support large parallelism through wavelength multiplexing[34,35]. These hardware-level advantages, however, do not automatically translate into system-level efficiency on programmable analog hardware, as analog–digital conversion overhead and repeated weight programming can dominate the total energy consumption and execution time unless the same configuration is reused across many operations[36]. Programmable analog Fourier processing therefore requires not only efficient physical hardware, but also hardware-aware algorithms that map the computation efficiently onto the processor, for example, by reusing programmed kernels across many operations.

Here, we introduce a hardware-aware stagewise radix-k FFT algorithm that reorders multidimensional FFT execution stage-by-stage across dimensions. The stagewise schedule preserves the same number of matrix–vector multiplications (MVMs) as conventional dimensionwise FFT execution while greatly reducing the number of physical weight reconfigurations. This reduction is achieved by applying each programmed FFT-stage kernel across the multidimensional data tensor before the processor is reconfigured. We implement this approach on a programmable analog-photonic matrix–vector multiplier and demonstrate it on automotive radar, a practically relevant application in which the input signals are available in analog form before digitization. In contrast to prior fixed-function analog FFT implementations, the Fourier transform is not hardwired into the circuit topology, but realized through programmable matrix–vector multiplications, allowing the transform size and dimensionality to be set algorithmically. We demonstrate radar processing on a 128×128 input data tensor and analyze the scaling of the stagewise schedule to larger sizes. Figure 1 summarizes the radar processing signal chain. Transmitted chirped signals are reflected by objects and converted into beat signals that encode target range, velocity, and angle (Fig. 1a,b). Extracting this information requires multidimensional Fourier transforms, which are decomposed into repeated MVMs and mapped efficiently to analog-photonic hardware (Fig. 1c). The resulting computation yields range–velocity maps that encode the target distance d and relative velocity Δv with respect to the radar sensor, as shown in Fig. 1d.

Results

Co-design FFTs for analog processors

In algorithm design, operation-count scaling is a primary measure of computational complexity. On digital hardware, efficient execution is further enabled by processor design techniques such as pipelining, caching, and branch prediction[37]. In programmable analog MVM processors, efficient execution also depends on how often a programmed processor state is reused before it must be updated. Increasing reuse lowers the effective energy per operation by amortizing programming overhead over many computations[36].

FFTs are particularly relevant here because the Cooley–Tukey algorithm computes them as stages of recurring butterfly operations[3]. In higher dimensions, FFTs are typically executed dimensionwise as successive one-dimensional transforms along each axis. On reconfigurable analog hardware, however, this ordering is inefficient because the identical butterfly weights are repeatedly loaded for different tensor slices instead of being reused while they remain programmed. Here, we instead introduce a stagewise schedule, applying all butterflies of a given stage across the full multidimensional data tensor before reconfiguring the processor.

More specifically, in the standard Cooley–Tukey radix-k FFT, each stage applies recurring $k \times k$ kernel blocks to different subsets of the data, see Fig. 2a. These kernel blocks are referred to as butterfly operations. In the radix-2 case considered here, each butterfly is represented by the 2×2 complex-valued matrix multiplication

$$\hat{W}_N^m \cdot \vec{y} \coloneqq \begin{pmatrix} 1 & W_N^m \\ 1 & -W_N^m \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \tag{1}$$

where $W_N^m = \mathrm{e}^{-\mathrm{i}2\pi\frac{m}{N}}$ denotes the twiddle factor and y_1 and y_2 are the complex input samples of the butterfly. Existing multidimensional FFT schedules, including row-column and vector-radix algorithms[4,38], preserve the characteristic $N \log N$ operation-count scaling but are not designed to maximize reuse of programmed analog kernel configurations.

Formally, the multidimensional Cooley–Tukey schedule is reordered from dimensionwise execution to stagewise execution,

$$\bigotimes_{i=1}^{d} \left(\prod_{j=1}^{\log_2 N} S_{i,j} \right) = \prod_{j=1}^{\log_2 N} \left(\bigotimes_{i=1}^{d} S_{i,j} \right), \tag{2}$$

where $S_{i,j}$ denotes the kernel block of stage j in dimension i. The left-hand side represents the conventional dimensionwise schedule, whereas the right-hand side represents the stagewise schedule. This reordering of the Kronecker product preserves the transform[39] while increasing the reuse of programmed stage configurations; the derivations are provided in the Supplementary Information.

We quantify the different execution schedules by the reuse factor R, defined as the average number of MVM executions per programmed kernel configuration. Table 1 compares dimensionwise, stagewise, and vector-radix schedules in terms of kernel size, MVM count, reprogrammings, and reuse R. Stagewise execution is identical to dimensionwise execution for $d = 1$, whereas for multidimensional transforms, it preserves the MVM count but reduces reprogramming overhead. Consequently, Fig. 2b shows that R increases strongly with dimensionality d and input size N, reaching $R > 100$ for two-dimensional transforms with $N \geq 32$.

The expressions shown in Table 1 are given for radix-2 FFTs. In practice, analog hardware typically supports larger kernel sizes, enabling higher-radix implementations with fewer total MVMs. Although the reuse factor decreases with increasing kernel size, Fig. 2c shows that for $N \geq 64$ it remains above $R = 100$, allowing substantially higher reuse of programmed kernel configurations.

Implementation on an analog-photonic processor

We implement the stagewise FFT algorithm on an existing analog-photonic matrix–vector multiplier[40]. The processor is based on a silicon-photonic crossbar array of size 9×3 and supports

arbitrary real-valued matrix–vector multiplications, see Fig. 3a. In this architecture, input values are represented by modulated optical powers, while matrix weights are implemented as tunable transmissivities controlled by electro-absorption modulators. An FPGA-based control unit together with digital-to-analog converters (DACs) and analog-to-digital converters (ADCs) provides the electrical drive signals for programming and data injection.

Because the photonic processor implements only real-valued MVMs, each complex radix-2 butterfly is first rewritten in terms of its real and imaginary components, yielding a logical 4×4 kernel. The complex-valued butterfly operation from Eq. (1) translates into its real-valued equivalent

$$\hat{W}_N^m \cdot \vec{y} \xrightarrow{\mathbb{R}} \begin{pmatrix} 1 & 0 & \mathrm{Re}(W_N^m) & -\mathrm{Im}(W_N^m) \\ 0 & 1 & \mathrm{Im}(W_N^m) & \mathrm{Re}(W_N^m) \\ 1 & 0 & -\mathrm{Re}(W_N^m) & \mathrm{Im}(W_N^m) \\ 0 & 1 & -\mathrm{Im}(W_N^m) & -\mathrm{Re}(W_N^m) \end{pmatrix} \cdot \begin{pmatrix} \mathrm{Re}(y_1) \\ \mathrm{Im}(y_1) \\ \mathrm{Re}(y_2) \\ \mathrm{Im}(y_2) \end{pmatrix}. \tag{3}$$

In the present demonstration, however, this logical kernel is not programmed as a single 4×4 matrix, but executed using 4×1 tiles, which define the effective compute unit of our experimental setup, see Supplementary Information. The full butterfly is therefore assembled from four sequential tile operations corresponding to the four rows of the expanded kernel. Intermediate outputs are stored temporarily in a local copy of the data array, because the remaining tile operations of the same logical butterfly still require access to the original input values.

For a given FFT stage, each programmed tile configuration corresponds to a fixed physical weight state of the photonic processor. This weight state is applied to all corresponding tensor slices before the next configuration is loaded, so that each butterfly block is reused across the data tensor before reprogramming. This execution order directly realizes the stagewise schedule on hardware. Since dimensionwise and stagewise schedules require the same number of MVMs, timing differences arise from the number of physical weight reconfigurations. In the present setup, where weight updates are slow compared with MVM execution, the reduced reconfiguration count of the stagewise schedule also lowers the measured configuration-time overhead, as shown in the Supplementary Information.

A practical limitation of the present analog-photonic FFT implementation is the finite 8-bit precision of the DAC and ADC interfaces used for weight programming and readout. Additional noise and hardware nonidealities further limit the effective twiddle accuracy. We therefore quantify the twiddle-weight error as the relative deviation between programmed and target twiddle weights and derive a corresponding effective number of bits (ENOB), see Methods.

In the present setup, we measure an average relative twiddle-weight error of $2.78\ \%$, corresponding to an effective weight setting precision of 4.88 bits. This is below the nominal 8-bit limit, reflecting additional hardware imperfections beyond digital quantization. Figure 3b shows the mapping from target to measured weights. Because each complex twiddle factor is decomposed into real and

imaginary components, the resulting weight accuracy depends on the precision of the corresponding hardware channel. The two channels exhibit imbalanced errors, so the effective twiddle-weight error differs between the even and odd rows of Eq. (3).

We first validate the hardware implementation in controlled one-dimensional experiments. As a representative input, we use a chirped signal sampled with $N = 64$ points, see Fig. 3c. This choice is loosely motivated by the later radar application, but does not affect the generality of the validation. The input is processed by the stagewise FFT on the photonic hardware and compared with the corresponding digital FFT, see Fig. 3d.

The measured error increases with the number of sample points, see Fig. 3e, consistent with the accumulation of finite-precision twiddle and MVM errors over a growing number of butterfly stages. This behavior is expected for this analog hardware implementation, because finite weight precision and readout noise accumulate across stages. Repeating each butterfly operation and averaging the outputs reduces noise contributions from the analog signal chain and thereby reduces the MVM error and consequently improves FFT precision, see also Supplementary Information. In the demonstrated case of $N = 64$, averaging over 2^8 repeated measurements yields an effective MVM error of $\sigma_{\mathrm{MVM}} = 5.5\ \%$ for individual 4×1 operations and reduces the global FFT magnitude error to $4.7\ \%$, see also Methods. In the present setup, such averaging is needed because the amplified spontaneous emission (ASE) light source introduces relatively high intensity noise. This averaging could be significantly reduced by using a lower-noise incoherent source, for example, separate wavelengths for different hardware channels[35,40].

Photonic processing of automotive radar data

Automotive radar systems rely on efficient range–velocity estimation of moving objects in real time, making multidimensional Fourier processing a central computational task[5]. Frequency-modulated continuous-wave (FMCW) radar is widely used for this purpose in the automotive industry[41]. In an FMCW radar, chirped electromagnetic signals are transmitted, reflected by objects, and mixed with the current transmitted signal at the receiver[6]. The resulting beat signal encodes target distance through the propagation delay, while repetition over multiple chirps enables relative velocity estimation. This signal structure maps onto a two-dimensional Fourier transform, in which the sampled points within a chirp define the range bins, and the repeated chirps define the velocity bins. Because these radar beat signals exist in the analog domain before digitization, FMCW radar is a natural candidate for analog or near-sensor processing[41]. In addition, silicon-photonic radar receivers have already been demonstrated, highlighting photonics as a compatible platform for integrated radar signal acquisition and processing[42,43].

We demonstrate the stagewise 2D FFT in this setting using a single isolated moving target. In our

experiment, a target generator applies a controlled delay and frequency shift to emulate an object at defined range and velocity before returning the modified signal to the radar sensor (Fig. 4a). In the radar experiment, the range–velocity map is obtained from a 128×128 input data matrix comprising 128 samples per chirp and 128 chirps. Mixing the transmitted signal T_x and the received signal R_x yields the beat signal, which is processed by the photonic 2D FFT to obtain a range–velocity map. Because the photonic system is limited by 8-bit analog-to-digital conversion, its dynamic range is on the order of $50\ \mathrm{dB}$ (Eq. (7)). To allow a fair comparison, both range–velocity maps are normalized to the same dynamic range. The range–velocity map shown in Fig. 4b exhibits two prominent features. A strong signal near $v = 0$, which decreases with increasing range, originates from back-reflections of static objects throughout the radar scene. A weaker, localized signal appears at approximately $v \approx -10\ \mathrm{m\,s^{-1}}$ and $d \approx 5\ \mathrm{m}$, corresponding to the emulated moving object. The negative relative velocity indicates that the target range is decreasing, consistent with an approaching target, as would occur for a preceding vehicle that is braking. We compare the resulting photonic map directly with a digital 2D FFT reference, as shown in Fig. 4c. We use peak localization as the key evaluation metric[2].

To quantitatively compare the digital and photonic computing results, we determine the dominant peak position by fitting sinc^2 profiles along the range and velocity axes of the range–velocity map. For an individual FMCW target, the beat signal is approximately sinusoidal, and the finite observation window results in a sinc^2-shaped intensity response, see also Methods. The results are shown in Fig. 4b,c. The resulting range estimates are $d_{0,\mathrm{digital}} = 4.89\ \mathrm{m}$ and $d_{0,\mathrm{photonic}} = 4.92\ \mathrm{m}$, while the corresponding velocity estimates are $v_{0,\mathrm{digital}} = -12.25\ \mathrm{m\,s^{-1}}$ and $v_{0,\mathrm{photonic}} = -12.21\ \mathrm{m\,s^{-1}}$. The digital and photonic range estimates differ by only $\Delta d_{\mathrm{bins}} = 0 = 0.03\ \mathrm{m}$, corresponding to 0.13 range bins, while the velocity estimates differ by $\Delta v = 0.04\ \mathrm{m\,s^{-1}}$, or $\Delta v_{\mathrm{bins}} = 0.16$ velocity bins. Despite the limited dynamic range and additional photonic noise sources, the photonic FFT therefore closely preserves the application-relevant range and velocity peak positions.

The importance of efficient radar signal processing is expected to grow further in modern automotive radar systems, where multiple-input multiple-output (MIMO) architectures with several transmitters and receivers additionally provide horizontal and vertical angle information[2]. This extends the processing from two-dimensional maps to higher-dimensional Fourier transforms, including 4D FFTs[7], which could benefit even more strongly from high weight reuse and parallel photonic processing. In such settings, reduced reprogramming overhead through stagewise scheduling could help to reduce the computational bottleneck and support more efficient radar processing in autonomous driving systems.

Discussion

Efficient analog processing depends not only on arithmetic workload, but also on hardware-aware scheduling of computation to minimize reprogramming overhead and maximize kernel reuse. We therefore introduce a stagewise radix-k FFT schedule that executes multidimensional FFTs stage-by-stage across dimensions. Implemented on an analog-photonic matrix–vector multiplier and demonstrated here on automotive FMCW radar, this approach achieves substantially higher reuse than row-column and vector-radix FFTs, with gains increasing with dimensionality. While no benefit arises in one dimension, the reuse factor is already $R > 100$ for two-dimensional transforms with input sizes of $N \geq 32$. As a result, the programming overhead per operation, E_{weight}/R, falls below $1\ \%$ of the original weight-programming cost E_{weight}. Importantly, this advantage persists even as hardware size and radix increase. These results identify structured kernel reuse as an important design principle for programmable analog processors.

More broadly, prior work has shown that hardware efficiency depends strongly on execution order and dataflow. In deep neural network accelerators, reuse-aware mappings reduce data movement and improve energy efficiency[31,44,45]. For analog hardware, this principle is even more important because repeated weight programming and analog–digital conversions can dominate the energy budget, making system efficiency critically dependent on kernel reuse[36,46]. This objective differs from prior stepwise FFT schedules[47] and from libraries such as FFTW[10], where the execution order is optimized primarily to reduce memory movement and improve data-cache locality, often favoring depth-first traversal. In reconfigurable analog-photonic processors, by contrast, the FFT coefficients are physical weight states, making a breadth-first, stagewise traversal advantageous because one programmed FFT-stage matrix is reused across many MVMs before reconfiguration. This weight-reuse perspective is also reflected in stationary-weight execution and related in-memory computing approaches[40,48,49]. In multi-core non-volatile in-memory processors, for example, different weight tiles can be stored across multiple cores and reused during inference, reducing the need to repeatedly reprogram a single compute array[22]. In our case, the reuse is achieved algorithmically by reordering multidimensional FFT execution on a single reconfigurable photonic processor. In this sense, we interpret the stagewise FFT as a reuse-aware mapping strategy for multidimensional transforms on programmable analog processors. Because the stagewise FFT schedule follows from the Kronecker-product structure of multidimensional operators, similar scheduling strategies can also be applied to Walsh–Hadamard transforms and more general Kronecker-structured linear-algebra algorithms[50].

References

1. Rao, K., Kim, D. & Hwang, J.-J. *Fast Fourier Transform - Algorithms and Applications*. Signals and Communication Technology (Springer Netherlands, Dordrecht, 2010). URL http://link.springer.com/10.1007/978-1-4020-6629-0.

2. Patole, S. M., Torlak, M., Wang, D. & Ali, M. Automotive radars: A review of signal processing techniques. *IEEE Signal Process. Mag.* **34**, 22–35 (2017). URL http://ieeexplore.ieee.org/document/7870764/.

3. Cooley, J. W. & Tukey, J. W. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comput.* **19**, 297–301 (1965).

4. Duhamel, P. & Vetterli, M. Fast fourier transforms: A tutorial review and a state of the art. *Signal Process.* **19**, 259–299 (1990). URL https://linkinghub.elsevier.com/retrieve/pii/016516849090158U.

5. Engels, F. *et al.* Automotive Radar Signal Processing: Research Directions and Practical Challenges. *IEEE J. Sel. Top. Signal Process.* **15**, 865–878 (2021). URL https://ieeexplore.ieee.org/document/9369027/.

6. Venon, A., Dupuis, Y., Vasseur, P. & Merriaux, P. Millimeter Wave FMCW RADARs for Perception, Recognition and Localization in Automotive Applications: A Survey. *IEEE Trans. Intell. Veh.* **7**, 533–555 (2022). URL https://ieeexplore.ieee.org/document/9760104/.

7. Fan, L. *et al.* 4D mmWave Radar for Autonomous Driving Perception: A Comprehensive Survey. *IEEE Trans. Intell. Veh.* **9**, 4606–4620 (2024). URL https://ieeexplore.ieee.org/document/10477463/.

8. Dick, R. P. *et al.* Research Challenges for Energy-Efficient Computing in Automated Vehicles. *Computer* **56**, 47–58 (2023). URL https://ieeexplore.ieee.org/document/10058815/.

9. Tang, P. T. P. DFTI—a new interface for Fast Fourier Transform libraries. *ACM Trans. Math. Softw.* **31**, 475–507 (2005). URL https://dl.acm.org/doi/10.1145/1114268.1114271.

10. Frigo, M. & Johnson, S. The Design and Implementation of FFTW3. *Proc. IEEE* **93**, 216–231 (2005). URL http://ieeexplore.ieee.org/document/1386650/.

11. Střelák, D. & Filipovič, J. Performance analysis and autotuning setup of the cuFFT library. 1–6 (ACM, Limassol, Cyprus, 2018). URL https://dl.acm.org/doi/10.1145/3295816.3295817.

12. Hameed, R. *et al.* Understanding sources of inefficiency in general-purpose chips. *Comput. Archit. News* **38**, 37–47 (2010).

13. Kuo, J.-C., Wen, C.-H., Lin, C.-H. & Wu, A.-Y. VLSI Design of a Variable-Length FFT/IFFT Processor for OFDM-Based Communication Systems. *EURASIP J. Adv. Signal Process.* 439360 (2003). URL https://asp-eurasipjournals.springeropen.com/articles/10.1155/S1110865703309060.

14. Maharatna, K., Grass, E. & Jagdhold, U. A 64-Point Fourier Transform Chip for High-Speed Wireless LAN Application Using OFDM. *IEEE J. Solid-State Circuits* **39**, 484–493 (2004). URL http://ieeexplore.ieee.org/document/1269925/.

15. Sadhu, B., Sturm, M., Sadler, B. M. & Harjani, R. Analysis and Design of a 5 GS/s Analog Charge-Domain FFT for an SDR Front-End in 65 nm CMOS. *IEEE J. Solid-State Circuits* **48**, 1199–1211 (2013). URL http://ieeexplore.ieee.org/document/6506133/.

16. Reshma P. G., Gopi, V. P., Babu, V. S. & Wahid, K. A. Analog CMOS implementation of FFT using cascode current mirror. *Microelectron. J.* **60**, 30–37 (2017). URL https://linkinghub.elsevier.com/retrieve/pii/S0026269216307212.

17. Yantir, H. E., Guo, W., Eltawil, A. M., Kurdahi, F. J. & Salama, K. N. An Ultra-Area-Efficient 1024-Point In-Memory FFT Processor. *Micromachines* **10**, 509 (2019). URL https://www.mdpi.com/2072-666X/10/8/509.

18. Zhao, H. *et al.* Energy-efficient high-fidelity image reconstruction with memristor arrays for medical diagnosis. *Nat. Commun.* **14**, 2276 (2023). URL https://www.nature.com/articles/s41467-023-38021-7.

19. Macfaden, A. J., Gordon, G. S. D. & Wilkinson, T. D. An optical Fourier transform coprocessor with direct phase determination. *Sci. Rep.* **7**, 13667 (2017). URL https://www.nature.com/articles/s41598-017-13733-1.

20. Zhu, H. H. *et al.* Space-efficient optical computing with an integrated chip diffractive neural network. *Nat. Commun.* **13**, 1044 (2022). URL https://www.nature.com/articles/s41467-022-28702-0.

21. Hu, J. *et al.* Diffractive optical computing in free space. *Nat. Commun.* **15**, 1525 (2024). URL https://www.nature.com/articles/s41467-024-45982-w.

22. Le Gallo, M. *et al.* A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference. *Nat. Electron.* **6**, 680–693 (2023). URL https://www.nature.com/articles/s41928-023-01010-1.

23. Ambrogio, S. *et al.* An analog-AI chip for energy-efficient speech recognition and transcription. *Nature* **620**, 768–775 (2023). URL https://www.nature.com/articles/s41586-023-06337-5.

24. Aguirre, F. *et al.* Hardware implementation of memristor-based artificial neural networks. *Nat. Commun.* **15**, 1974 (2024). URL https://www.nature.com/articles/s41467-024-45670-9.

25. Shastri, B. J. *et al.* Photonics for artificial intelligence and neuromorphic computing. *Nat. Photon.* **15**, 102–114 (2021). URL https://www.nature.com/articles/s41566-020-00754-y.

26. Zhou, W. *et al.* In-memory photonic dot-product engine with electrically programmable weight banks. *Nat. Commun.* **14**, 2887 (2023). URL https://www.nature.com/articles/s41467-023-38473-x.

27. Wu, B., Zhou, H., Dong, J. & Zhang, X. Programmable integrated photonic coherent matrix: Principle, configuring, and applications. *Appl. Phys. Rev.* **11**, 011309 (2024). URL https://pubs.aip.org/apr/article/11/1/011309/3261372/Programmable-integrated-photonic-coherent-matrix.

28. Ríos, C. *et al.* In-memory computing on a photonic platform. *Sci. Adv.* **5**, eaau5759 (2019). URL https://www.science.org/doi/10.1126/sciadv.aau5759.

29. Zhu, Y. *et al.* Silicon photonics convolution accelerator based on coherent chips with sub-1 pJ/MAC power consumption. *Photon. Res.* **13**, 497 (2025). URL https://opg.optica.org/abstract.cfm?URI=prj-13-2-497.

30. Ahmed, S. R. *et al.* Universal photonic artificial intelligence acceleration. *Nature* **640**, 368–374 (2025). URL https://www.nature.com/articles/s41586-025-08854-x.

31. Xu, Z. *et al.* Large-scale photonic chiplet Taichi empowers 160-TOPS/W artificial general intelligence. *Science* **384**, 202–209 (2024). URL https://www.science.org/doi/10.1126/science.adl1203.

32. Bernstein, L. *et al.* Single-shot optical neural network. *Sci. Adv.* **9**, eadg7904 (2023). URL https://www.science.org/doi/10.1126/sciadv.adg7904.

33. Hua, S. *et al.* An integrated large-scale photonic accelerator with ultralow latency. *Nature* **640**, 361–367 (2025). URL https://www.nature.com/articles/s41586-025-08786-6.

34. Hamerly, R. *et al.* Netcast: Low-Power Edge Computing with WDM-defined Optical Neural Networks. *Journal of Light. Technol.* **42**, 7795–7806 (2024). URL 10.1109/JLT.2024.3405799.

35. Feldmann, J. *et al.* Parallel convolutional processing using an integrated photonic tensor core. *Nature* **589**, 52–58 (2021). URL https://www.nature.com/articles/s41586-020-03070-1.

36. Bahr, N., Steinmeyer, D. & Pernice, W. Energy Efficiency in Analog Photonic Processors: Conversions and Losses at Scale. In *ISC High Performance 2026 Research Paper Proceedings (41st International Conference)*, 1–9 (IEEE, Hamburg, Germany, 2026). URL https://ieeexplore.ieee.org/document/11520497/.

37. Hennessy, J. L. & Patterson, D. A. *Computer Architecture: A Quantitative Approach* (Morgan Kaufmann Publishers, 2019), 6 edn.

38. Harris, D., McClellan, J., Chan, D. & Schuessler, H. Vector radix fast Fourier transform. In *IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 2, 548–551 (1977). URL http://ieeexplore.ieee.org/document/1170349/.

39. Horn, R. A. & Johnson, C. R. *Matrix Analysis* (Cambridge University Press, 2012), 2 edn.

40. Meyer, L. *et al.* Deep neural network inference on an integrated, reconfigurable photonic tensor processor. *Nat. Commun.* **17**, 3396 (2026). URL https://www.nature.com/articles/s41467-026-71599-2.

41. Fei, T. Automotive Radar Systems: Architecture, Signal Processing, and Future Perspectives. In Koten, H. & Sarıkoç, S. (eds.) *Vehicle Technology and Automotive Engineering* (IntechOpen, 2025). URL https://www.intechopen.com/chapters/1217095.

42. Ghelfi, P. *et al.* A fully photonics-based coherent radar system. *Nature* **507**, 341–345 (2014). URL https://www.nature.com/articles/nature13078.

43. Kruse, S. *et al.* Silicon Photonic Radar Receiver IC for mm-Wave Large Aperture MIMO Radar Using Optical Clock Distribution. *IEEE Microw. Wireless Compon. Lett.* **32**, 1447–1450 (2022). URL https://ieeexplore.ieee.org/document/9832612/.

44. Parashar, A. *et al.* Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *IEEE Int. Symp. Perfrom. Anal. Syst. Softw.*, 304–315 (Madison, WI, USA, 2019). URL https://ieeexplore.ieee.org/document/8695666/.

45. Chen, Y.-H., Emer, J. & Sze, V. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *IEEE International Symposium on Computer Architecture*, 367–379 (Seoul, South Korea, 2016). URL http://ieeexplore.ieee.org/document/7551407/.

46. Zhang, C., Yuan, Z., Li, X. & Sun, G. Algorithm-Hardware Co-Design for Energy-Efficient A/D Conversion in ReRAM-Based Accelerators. In *IEEE Des. Autom. Test Eur. Conf. Exhib.*, 1–6 (Valencia, Spain, 2024). URL https://ieeexplore.ieee.org/document/10546760/.

47. Anderson, G. A stepwise approach to computing the multidimensional fast Fourier transform of large arrays. *IEEE Trans. Signal Process.* **28**, 280–284 (1980). URL http://ieeexplore.ieee.org/document/1163404/.

48. Leroux, N. *et al.* Analog in-memory computing attention mechanism for fast and energy-efficient large language models. *Nat. Comput. Sci.* **5**, 813–824 (2025). URL https://www.nature.com/articles/s43588-025-00854-1.

49. Kösters, D. J. *et al.* Benchmarking energy consumption and latency for neuromorphic computing in condensed matter and particle physics. *APL Mach. Learn.* **1**, 016101 (2023). URL https://pubs.aip.org/aml/article/1/1/016101/2878720/Benchmarking-energy-consumption-and-latency-for.

50. van Loan, C. F. The ubiquitous Kronecker product. *J. Comput. Appl. Math.* **123**, 85–100 (2000). URL https://linkinghub.elsevier.com/retrieve/pii/S0377042700003939.

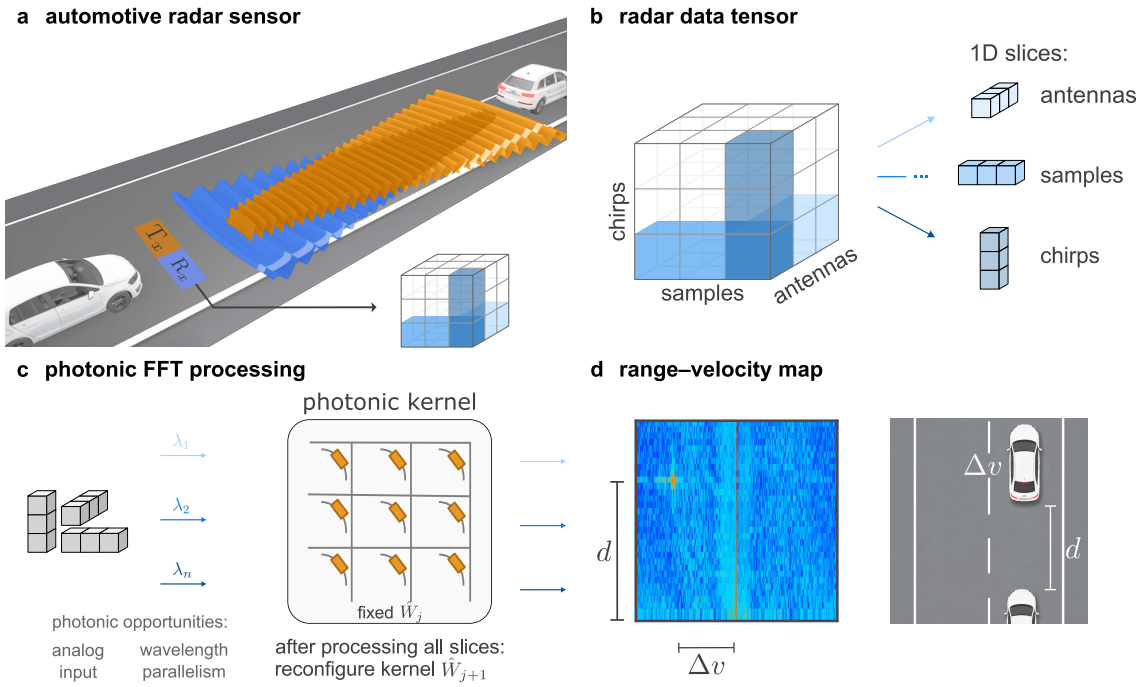

Figure 1: **Concept of analog-photonic radar signal processing. a,** Automotive FMCW radar transmits chirped signals T_x, that are reflected by objects and received as echoes R_x. **b,** Sampled signals of multiple chirps and antennas encode target range, relative velocity, and angle. Each frame is allocated in a tensor. **c,** Extracting physical target information requires multidimensional Fourier processing. The FFTs can be decomposed into repeated butterfly operations, which correspond to repeated MVMs that can be executed efficiently on programmable analog-photonic processors. The stagewise schedule maps this computation to the hardware by applying each programmed FFT-stage kernel across many tensor slices before reconfiguration, thereby reducing weight-programming overhead. **d,** The processed data yield a range–velocity map containing the relevant target information.

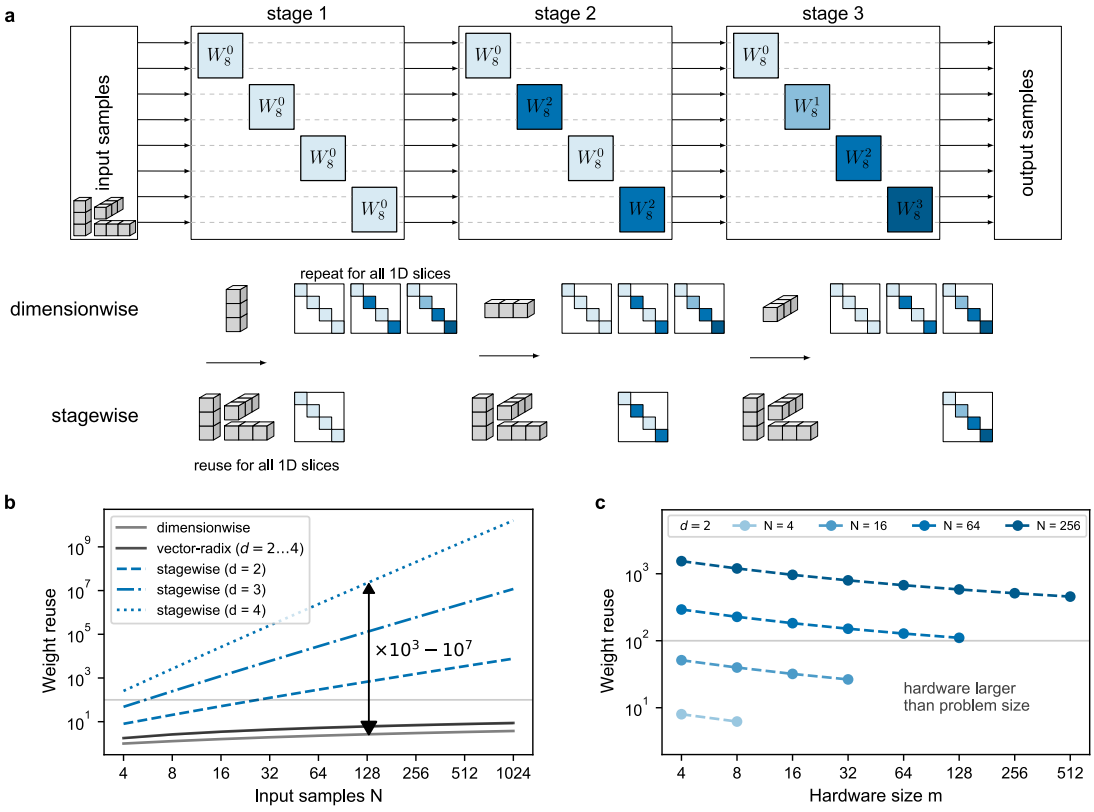

Figure 2: **Stagewise radix-2 FFT algorithm for analog processors. a,** Three stages of an 8-point radix-2 Cooley–Tukey FFT. Each stage can be written as a block-diagonal matrix with butterfly operators $\hat{W}_N^m$ on its diagonal. For a given stage, the same butterfly operators appear repeatedly, allowing a programmed hardware kernel to be reused before a new configuration is loaded. The stagewise schedule exploits this repetition by applying each butterfly operator $\hat{W}_N^m$ of a given stage across all one-dimensional slices of the multidimensional input tensor before proceeding to the next stage. In the dimensionwise schedule, by contrast, the full FFT of one slice is completed before moving to the next slice. **b,** Average weight reuse R quantifies how many MVMs are executed per programmed butterfly configuration before the hardware is reconfigured. For stagewise scheduling, R increases with input size N and transform dimensionality d. The horizontal line marks $R = 100$. **c,** Average weight reuse R for two-dimensional stagewise FFTs as a function of hardware size. Larger hardware kernels reduce the number of required butterfly configurations, but also reduce the number of repeated applications per individual configuration. Even in this case, the reuse remains above $R = 100$ for problem sizes $N \geq 64$.

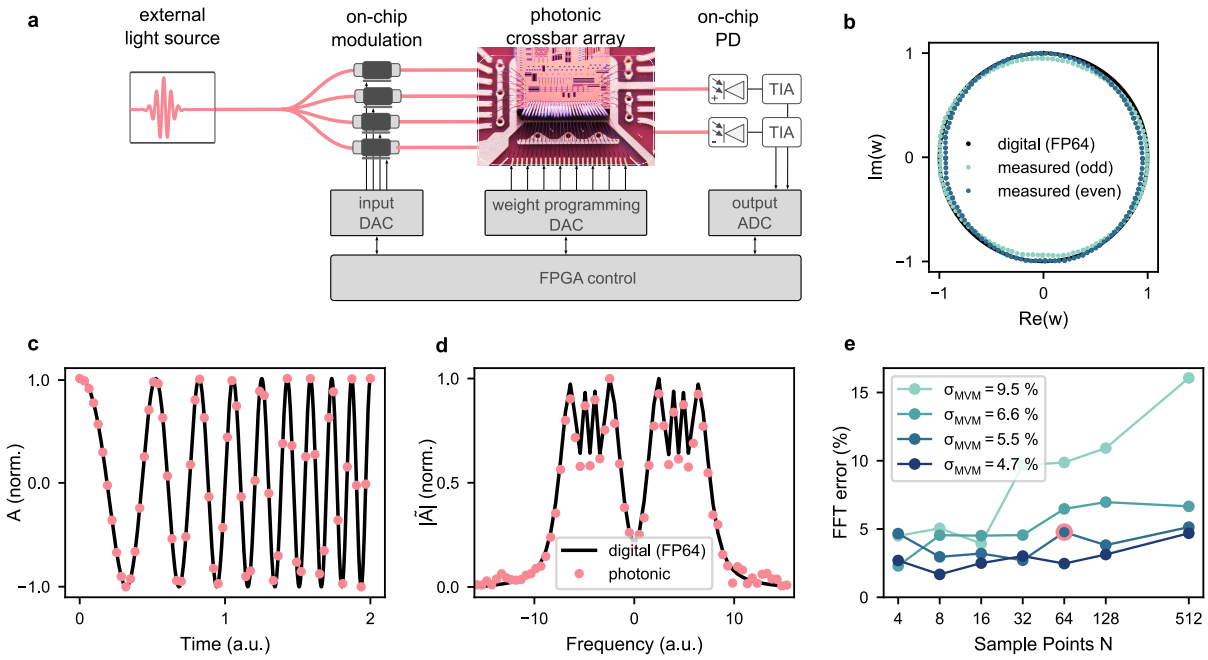

Figure 3: **Stagewise FFT on an integrated photonic processor. a,** Experimental setup of the FPGA-controlled silicon-photonic matrix processor. **b,** Comparison of digital FP64 target values and measured analog-photonic values for the twiddle-factor weights. The measured weights deviate from the ideal twiddle factors by $2.78\ \%$, corresponding to ENOB $= 4.88$ bits. **c,** Chirped input signal sampled with $N = 64$ points. **d,** Digital and photonic FFT results for the chirped input. **e,** Dependence of FFT error σ_{FFT} on the effective MVM error σ_{MVM} and transform size N. Different values of σ_{MVM} are obtained by averaging repeated measurements of each MVM operation, see also Supplementary Information. The circled marker indicates the parameters used in **c, d**.

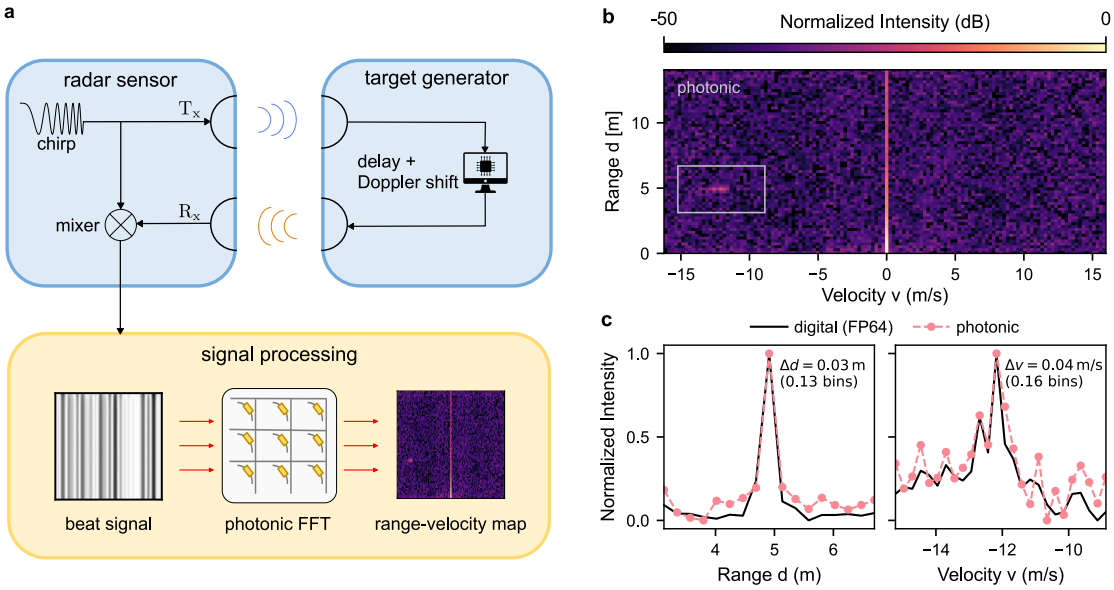

Figure 4: **Photonic 2D FFT processing for automotive radar. a,** Experimental setup for photonic radar signal processing. A transmitted chirp is received by a target generator, which applies a delay and Doppler shift to emulate an object at a chosen range and velocity before returning the signal to the radar sensor. Mixing T_x and R_x yields the beat signal, which is processed by a photonic 2D FFT to obtain the range–velocity map. **b,** Range–velocity intensity map obtained using photonic 2D FFT processing. **c,** Target range and velocity peak positions, extracted by sinc^2 fitting, are closely preserved after photonic processing.

	Dimensionwise	Stagewise (this work)	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 1: **Comparison of multidimensional FFT schedules for radix-2 butterflies.** Dimensionwise and stagewise execution require the same number of MVMs, but differ in the number of kernel reprogrammings. This results in much higher reuse for the stagewise schedule. Vector-radix exploits multidimensional structure but does not achieve the same reuse scaling.

Author contributions: Conceptualization: NB, FB, DS, WP
Methodology: NB, TG, FO, JD, LM, FE, LK
Investigation: NB
Visualization: NB
Project administration: DS, WP
Supervision: FB, DS, WP
Writing - original draft: NB
Writing - review & editing: All authors

Competing interests: The authors declare that they have no competing interests.

Data availability: The data that support the findings of this study may be available from the first author and the corresponding author on reasonable request.

Methods

Discrete Fourier transform

For a sampled time-domain signal $A = (A_0, \ldots, A_{N-1})$, the discrete Fourier transform $\tilde{A} = (\tilde{A}_0, \ldots, \tilde{A}_{N-1})$ is defined as

$$\tilde{A}_i = \sum_{j=0}^{N-1} \exp\left(-\mathrm{i}2\pi \frac{ij}{N}\right) A_j = \sum_{j=0}^{N-1} W_N^{ij} A_j, \tag{4}$$

where $W_N^m = \exp(-\mathrm{i}2\pi m/N)$ denotes the twiddle factor.

Experimental setup

The system used for photonic MVMs has been described previously[40]. The photonic integrated circuit comprises a 9×3 optical crossbar array fabricated by imec on a silicon-on-insulator platform, with electro-absorption modulators for input and weight modulation, and on-chip photodiodes for readout. The system is controlled by an SoC FPGA (Xilinx ZCU216), while the weights are programmed using a slower multi-channel DAC. Optical input is supplied by an external amplified spontaneous emission source (Agilent 83438A). To reduce the intrinsic noise of this source, each computation is repeated multiple times and the averaged output is used as the result.

Weight programming error

The relative weight programming error is quantified as the normalized Euclidean deviation between the programmed and target complex twiddle weights,

$$\sigma_w = \frac{|w_{\mathrm{target}} - w_{\mathrm{measured}}|}{|w_{\mathrm{target}}|}. \tag{5}$$

When reporting a single value for the weight programming error, we use the average of σ_w over all programmed twiddle weights. Based on this quantity, the peak signal-to-noise ratio (PSNR) is defined as

$$\mathrm{PSNR} = -20 \log_{10} \sigma_w, \tag{6}$$

and the corresponding effective number of bits (ENOB) is calculated as

$$\mathrm{ENOB} = \frac{\mathrm{PSNR} - 1.76}{6.02}. \tag{7}$$

MVM error metric

The MVM error is quantified as the L_2 error relative to the digital reference

$$\sigma_{\mathrm{MVM}} = \frac{\langle \| x_{\mathrm{target},i} - x_{\mathrm{measured},i} \|_2 \rangle_i}{\langle \| x_{\mathrm{target},i} \|_2 \rangle_i} \tag{8}$$

where $x_{\mathrm{target},i}$ and $x_{\mathrm{measured},i}$ denote the digital-reference and measured output vectors of the i-th MVM operation. The values reported here are obtained from 1000 randomly generated 4×1 MVM operations.

FFT error metric

Before evaluating the error, the target and measured spectra are independently normalized to unit peak magnitude $|\hat{A}_i| := |\tilde{A}_i| / \max |\tilde{A}|$. FFT error is then evaluated against the digital reference as the root-mean-square difference between magnitudes of the normalized FFT coefficients $\tilde{A}_i$

$$\sigma_{\mathrm{FFT}} = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} \Big| \, |\hat{A}_{\mathrm{target},i}| - |\hat{A}_{\mathrm{measured},i}| \, \Big|^2}. \tag{9}$$

Radar experiment

We perform the automotive radar experiment using an INRAS 77 GHz FMCW radar sensor together with a 76–81 GHz transceiver (National Instruments mmRH-3608) that emulated an object at a defined distance and velocity. The resulting radar data are processed using both the photonic stagewise FFT and a digital reference implementation. Application-level accuracy is evaluated by comparing the dominant peak positions in the resulting range–velocity maps.

Peak-position and peak-velocity errors

For an individual FMCW target, the beat signal is approximately sinusoidal. The finite observation window results in a sinc^2-shaped intensity response in the Fourier domain following

$$I(x) = A \cdot \mathrm{sinc}^2 \left(\frac{x - x_0}{w} \right) + c, \tag{10}$$

where A denotes the peak amplitude, c a constant offset, w the peak-width parameter, and x_0 the fitted peak position. The fitted parameter x_0 is therefore used to estimate the target range or velocity. The deviation between the digital and photonic peak positions is calculated as

$$\Delta x = \left| x_{0,\mathrm{digital}} - x_{0,\mathrm{photonic}} \right|. \tag{11}$$

To express this deviation relative to the discrete FFT sampling, it is normalized by the corresponding range or velocity bin spacing δx_{bin} (see also Supplementary Information),

$$\Delta x_{\mathrm{bins}} = \frac{\Delta x}{\delta x_{\mathrm{bin}}}. \tag{12}$$

Here, $\Delta x_{\mathrm{bins}} < 1$ indicates that the difference between the digital and photonic peak estimates is smaller than one FFT bin.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- SupplementaryInformation.pdf