SparseDSP: System-Level Evaluation of Deterministic Sparse FFT for Radar, Sonar, and LiDAR

Aaron R. Flouro and Shawn P. Chadwick, PhD.
SparseTech, Iowa, USA
research@sparse-tech.com

SparseTech Technical Papers

***Abstract*—We present SparseDSP, a regime-adaptive deterministic sparse FFT system evaluated against Dense FFT across radar, sonar, electronic warfare, and LiDAR operating points. SparseDSP estimates signal sparsity internally, then dispatches to an exact engine selected by its internal dispatch policy from an internal family of deterministic sparse algorithms spanning $O(k \log k)$ to $O(k \log N)$ to $O(\sqrt{N} \log k)$, with Dense FFT as a fallback path. No external sparsity parameter is required. Validation follows three paths: Path A synthetic benchmark (4992 configurations, 100% exact recovery, median speedup 3.93× at low sparsity); Path B impairment robustness under off-grid, fading, and clutter-like conditions (≥99.44% exact across six channel models, versus 0.00–0.28% for Dense top-k heuristic); and Path C real-payload validation across six public datasets (argos, argoverse2, deepmimo, nexrad, nuscenes, oxford_radar), all achieving exact recovery (1.0). At selected sensing scales ($k = 10$), SparseDSP achieves 4.05× speedup at $N = 16$K, 4.87× at $N = 524$K, and 38.38× at $N = 4$M over Dense FFT. SparseDSP does not outperform Dense FFT in all regimes: at high support sizes ($k = 150$–200), SparseDSP achieves 100% exact recovery where Dense top-k selection does not (Dense exact rate 0% at these operating points). SparseDSP incurs higher wall-clock cost (median 497.9 ms vs Dense 24.1 ms) but produces correct output. We discuss potential relevance to automotive FMCW radar, airborne AESA, naval phased array, submarine and ASW sonar, synthetic aperture radar, electronic warfare spectrum surveillance, and FMCW LiDAR. All Path A/B/C results are measured on the stated x86 platform. Domain mappings to radar, sonar, SAR, EW, and LiDAR are literature-grounded workload interpretations, not deployment validations.**

I. Introduction

A. Active Sensing DSP Challenges

Modern active sensing systems across defense, commercial, and scientific domains face a common computational challenge: the growing gap between sensor bandwidth and available processing power. Radar, sonar, and LiDAR systems share a signal processing pipeline that begins with frequency-domain analysis of the received signal, whether the target quantity is range (FMCW beat frequency), velocity (Doppler shift), or emitter frequency (ESM/ELINT).

The computational cost of the core FFT step scales as $O(N \log N)$ where N is the signal length. As sensor bandwidths increase, N grows correspondingly:

- **77 GHz automotive FMCW radar**: up to 4 GHz RF sweep bandwidth (77–81 GHz band), $N \approx 1$K beat-signal samples per chirp, aggregating to ~ 1M samples per coherent data cube [1]
- **Airborne AESA radar**: Wideband pulse compression, $N = 262$K–1M range samples per CPI; beam dwells range from sub-millisecond (fast track updates) to several milliseconds (search) [2]
- **Naval phased array radar**: Simultaneous search and track, $N = 131$K–524K per dwell [3]
- **Submarine hull-mounted sonar**: Long CPI at 25 kHz, $N = 262$K–1.5M [4]
- **Towed-array sonar**: 30–60 s CPI at $f_s = 25$ kHz, yielding $N = 750$K–1.5M samples per channel [5]
- **ESM/ELINT receivers**: 2–18 GHz frequency coverage; at 1–2 GS/s digitization, a 0.5–1 ms capture produces $N = 0.5$M–2M samples [6]
- **Spaceborne SAR**: Full-aperture azimuth, $N = 1$M–4M per line [7]
- **FMCW LiDAR**: coherent detection with sweep bandwidths up to 200 MHz; per-pixel sample counts are architecture-dependent (e.g., $\sim 10^5$–10^6) [8]

In all these systems, the number of targets k (range bins containing returns, Doppler lines with targets, or emitter frequencies) is orders of magnitude smaller than N. This sparsity ($k \ll N$) is the structural property that sparse FFT algorithms exploit.

B. The Computational Gap in Numbers

To understand why sparse transforms become relevant at modern sensing scales, consider the FFT cost growth across signal lengths. Table I presents the dense FFT workload at representative operating points.

At legacy pulse-Doppler scales ($N \approx 1$K), the FFT is a negligible fraction of the processing budget. At modern FMCW, AESA, and sonar scales ($N \geq 65$K), the FFT step alone can consume a substantial share of the per-dwell or per-chirp latency budget. At SAR and ESM scales ($N \geq 2$M), dense FFT dominates the processing chain.

SparseDSP at $k = 10$ shifts the leading-order algorithmic dependence from $O(N \log N)$ to sublinear in N. The savings

Inquiries regarding evaluation access may be directed to research@sparse-tech.com.

TABLE I
DENSE FFT COST GROWTH AT SENSING SCALES

System	N	Complexity ($N \log_2 N$)	Budget pressure
Legacy pulse-Doppler	1K	10K	Low
Short-CPI sonar	16K	229K	Manageable
FMCW radar	65K	1.05M	Increasing
AESA / naval	262K	4.72M	Strained
SAR azimuth / ESM	2M	42M	Can exceed budget
Full-aperture SAR	4M	88M	Multi-core required

FLOPs approximate $5N \log_2 N$ for complex radix-2 FFT.

grow with signal length: at $N = 4$M, the measured speedup is 38.38× on the benchmark platform (Table VII).

C. Dense vs Sparse Sensing Pipelines

The following comparison illustrates the data-flow difference for a representative FMCW radar or sonar CPI scenario ($N = 262$K, $k = 10$):

Dense pipeline (conventional):

1) ADC captures $N = 262$K complex samples (chirp beat signal or sonar CPI)
2) Full N-point FFT: ≈ 4.72M FLOPs, produces N frequency bins
3) All N bins passed to downstream detector (CFAR, threshold, or tracker)
4) Detector selects k target bins from N candidates
5) Total compute: $O(N \log N)$; downstream receives N bins

Sparse pipeline (SparseDSP):

1) ADC captures $N = 262$K complex samples
2) SparseDSP estimates k internally, recovers $k = 10$ dominant bins directly
3) Only k results passed to downstream detector
4) Total compute: sublinear in N; downstream receives k bins

For $k = 10$ at $N = 262$K: the dense pipeline produces 262,144 frequency bins and lets the CFAR detector discard 262,134 of them. The sparse pipeline produces 10 bins directly. The computational and downstream data-flow difference grows with N.

This pipeline comparison is schematic. SparseDSP replaces only the transform stage; downstream detection, tracking, and classification stages remain unchanged. In some legacy processing chains, downstream detectors may expect a full N-bin spectrum and would require interface adaptation to accept k-bin sparse output. The benchmark task evaluates transform-stage bin identification, not end-to-end detection-theoretic performance.

D. Barriers to Sparse FFT Adoption in Sensing

Three factors may help explain the limited adoption of sparse FFT in production radar, sonar, and EW systems despite two decades of academic progress:

1) **Wrong scale**: Conventional pulse-Doppler radar typically uses Doppler CPI/FFT lengths from tens to ∼1K pulses [9], well below the sparse/Dense crossover at $N \approx 32$K. At these scales, sparse engine overhead exceeds the full FFT cost.
2) **Probabilistic guarantees**: The MIT sFFT family [10], [11] provides probabilistic bounds ($1 - \delta$ success), less aligned with deterministic assurance requirements in safety-critical contexts (ISO 26262 ASIL-D for automotive [12], DO-178C DAL-A for airborne [13]) and military requirements (MIL-STD-461G [14]).
3) **Implementation gap**: Prior implementations were not evaluated under a comparable deterministic validation protocol across thousands of configurations.

Table II compares prior work with SparseDSP.

TABLE II
PRIOR ART COMPARISON

Method	Complexity	Determ.	Validated
Dense FFT	$O(N \log N)$	Yes	Baseline
MIT sFFT 1.0 [10]	$O(\sqrt{Nk} \log N)$	No	Limited
MIT sFFT 2.0 [11]	$O(k \log N)$	No	Limited
Li–Caire CRT [15]	$O(k \log^2 N)$	Yes	Partial
SparseDSP	$O(k \log k)$–$O(\sqrt{N} \log k)$	Yes	This work

E. Scope and Framing

This paper is a *systems evaluation* that validates SparseDSP, a regime-adaptive deterministic sparse FFT system, against Dense FFT across operating points relevant to radar, sonar, electronic warfare, and LiDAR. It is not a theoretical contribution (the underlying algorithms are presented in companion papers [16]–[18]) and not a shootout against external implementations.

SparseDSP contains multiple deterministic sparse FFT engines that are dispatched based on signal parameters. SparseDSP treats the dispatch policy and per-engine selection boundaries as internal system parameters, focusing evaluation on externally observable aggregate performance. This paper reports the system-level evaluation methodology, the demonstrated exactness across 4992 synthetic configurations and six real-payload datasets, and the domain parameter mapping that connects abstract (N, k) benchmarks to physical sensing systems.

A companion report applies the same framework to 5G/6G wireless spectrum sensing [19]. The benchmark SDK is a domain-agnostic tool; it was applied here to multiple sensing domains using domain-specific parameter mappings.

SparseTech Technical Papers

SparseTech Technical Papers

F. Claim Scope

The claims in this paper fall into three categories. First, *validated empirical claims*: exactness and timing results measured on the stated x86 platform under the Path A/B/C protocols. Second, *domain mapping claims*: the correspondence between benchmark parameters (N, k) and representative sensing workloads drawn from radar, sonar, EW, and LiDAR literature. Third, *deployment interpretation claims*: discussion of where the observed complexity ratios may be operationally relevant. Only the first category is directly benchmark-validated in this paper. The second is literature-grounded interpretation, and the third is engineering extrapolation subject to platform-specific re-measurement.

TABLE III
CLAIM TAXONOMY

Claim	Evidence tier
Exact recovery across 4992 configs	Measured (Path A)
Impairment exactness $\geq$99.44%	Measured (Path B)
Real-payload exactness (1.0)	Measured (Path C)
Speedup ratios vs Dense FFT	Measured (x86)
FFTW baseline near-parity	Measured (x86)
(N, k) mapping to FMCW/AESA/ sonar/SAR/EW/LiDAR	Literature interpretation
Platform deployment feasibility	Not claimed
End-to-end CFAR/tracker performance	Not claimed
Power/SWaP-C measurements	Not claimed
DSP/FPGA/ASIC timing	Not claimed

TABLE IV
EVIDENCE PATHS AND SUPPORTED CLAIMS

Path	Evidence type	Claim supported
A	Synthetic controlled	Transform-stage exactness and speedup across (N, k) regimes
B	Impairment-calibrated	Robustness under off-grid, fading, and clutter conditions
C	Curated real-payload	Spectral fidelity on real-world signal structures

G. Contributions

1) **SparseDSP system architecture**: Deterministic dynamic dispatch across a family of sparse FFT engines with Dense FFT fallback.
2) **Path A synthetic validation**: 4992 configurations (4032 low-k, 960 high-k), 100% exact recovery, with speedup distribution and explicitly reported high-k correctness regime.
3) **Path B impairment robustness**: System-level exactness under off-grid, fading, and clutter-like conditions across six channel models.
4) **Path C curated real-payload evaluation**: Six public datasets spanning radar, LiDAR, weather, MIMO, and autonomous driving domains, all achieving Dense-reference exact recovery on curated known-sparsity subsets.
5) **Domain parameter mapping**: Connecting benchmark results to radar, sonar, SAR, EW, and LiDAR operating points with timing at representative scales.

H. Illustrative Example

To ground SparseDSP's behavior, consider two concrete scenarios:

Scenario A ($N = 262$K, $k = 10$, spread placement):

SparseDSP dispatches to an exact engine selected by its internal dispatch policy. Result: SparseDSP completes in 1.173 ms versus Dense at 4.792 ms, a 4.18$\times$ speedup with identical frequency-bin output.

Scenario B ($N < 32$K, moderate k):

At legacy pulse-Doppler scales with moderate sparsity, Dense FFT is typically preferred because sparse engine overhead approaches or exceeds FFT cost. The crossover is k-dependent: at very low k (≤ 5), sparse engines can still win at small N, but production dispatch policies may conservatively prefer Dense in this regime as a practical safety margin.

Scenario C ($N = 524$K, $k = 175$, near-dense regime):

SparseDSP dispatches and produces exact output. Dense top-k does not produce exact output at $k = 175$: with many bins at similar energy levels, magnitude-based ranking frequently selects incorrect bins (Dense avg recall 0.7528 at $k = 150$–200, exact rate 0/960). SparseDSP incurs approximately 20$\times$ higher wall-clock cost but is the only option for exact recovery at this operating point.

II. SPARSEDSP SYSTEM ARCHITECTURE

A. Design Principles

SparseDSP satisfies three properties:

- **Deterministic**: Given identical inputs, SparseDSP always returns the same output via the same engine. No randomized dispatch or tie-breaking.
- **Composable**: SparseDSP can be embedded in any signal processing pipeline that provides a fixed-length input block. Sparsity is estimated internally; no external k parameter is required.
- **Fallback-based**: Every sparse dispatch includes a Dense FFT fallback path activated by post-detection verification failure.

B. Multi-Engine Architecture

SparseDSP contains a family of deterministic sparse FFT engines based on Chinese Remainder Theorem reconstruction, coprime-modulus subsampling, and structured keyed gating. Individual engines span $O(k \log k)$ to $O(k \log N)$ to $O(\sqrt{N} \log k)$, each suited to different parameter regions of the (N, k) space. The algorithms are presented in companion papers [16]–[18]. All engines are deterministic, require no randomization, and produce verifiable outputs.

SparseDSP does not require the sparsity level k to be supplied externally. For each input transform, the system

SparseTech Technical Papers

estimates the effective sparsity internally as part of its dispatch pipeline and then selects the corresponding deterministic sparse recovery procedure. A deterministic dispatch function maps the input block and estimated sparsity to an exact engine selected by its internal dispatch policy. Dense FFT serves as the fallback path. Dispatch overhead is $O(1)$ and measured to be small relative to transform cost at $N \geq 65$K. All dispatch parameters were fixed before benchmark execution and held constant across all reported experiments.

SparseDSP is stateless per transform: each processing cycle (e.g., a radar chirp or sonar CPI) is handled independently from a fresh fixed-length input block, and the system re-estimates the effective sparsity for that block before recovery. Changes in the number of targets across successive cycles are absorbed by the normal runtime estimation step rather than by external control logic or manual reconfiguration.

SparseDSP is designed to occupy the FFT stage as a replacement for a conventional FFT block within an existing sensing pipeline. The transform length N is determined upstream by the sensor configuration (e.g., ADC sample rate and chirp duration for radar, sample rate and CPI duration for sonar), and SparseDSP operates on the resulting fixed-length input block. It does not select waveform parameters, control data acquisition, or perform downstream detection stages such as CFAR.

C. Dense FFT Baseline

The Dense backend computes the full $O(N \log N)$ FFT via RustFFT 6.4.1 [20] followed by a top-k magnitude selection heuristic (select-nth-unstable-by on magnitude-sorted bins). This is *not* a matched detector or CFAR processor; it is a simple "return the k largest bins" operation. The benchmark evaluates bin-identification throughput under controlled sparsity, not detection-theoretic P_d/P_{fa} optimality.

The Dense FFT transform itself computes all N bins exactly. The failure at high k is in the top-k magnitude selection heuristic: with $k = 150$–200 targets, many bins have similar energy levels, and magnitude-based ranking frequently selects incorrect bins (Dense avg recall 0.7528 at $k = 150$–200, exact rate 0/960). SparseDSP's algebraic reconstruction identifies bins through residue structure rather than magnitude ranking, maintaining exact recovery regardless of amplitude distribution.

Plan reuse, buffer reuse, and black-box optimization barriers were applied consistently across implementations.

Why RustFFT: RustFFT was selected because it integrates directly with the SparseDSP benchmark harness under the same language (Rust), compiler (LLVM), precision (FP32), and memory-management model, reducing cross-library integration confounders.

FFTW calibration: To verify that RustFFT is comparable to FFTW on this platform, we performed a same-platform calibration against FFTW 3 (via pyFFTW 0.15.1) under controlled conditions: single-thread, P-core-only affinity, FP32, real-input (R2C) transforms, FFTW planner flag `FFTW_MEASURE`, planning time excluded, 32-byte SIMD-aligned allocation, warm cache after 5–10 untimed iterations, and 12–100 measured iterations per size. RustFFT used `realfft` 3 + `rustfft` 6.1 compiled with `-C target-cpu=native` (AVX2/FMA enabled). Table V presents the results.

TABLE V
DENSE FFT BASELINE CALIBRATION: RUSTFFT VS FFTW (R2C, SINGLE-THREAD, P-CORE, FP32)

N	RustFFT (ms)	FFTW (ms)	Relative
16K	0.0139	0.0124	FFTW 1.12×
65K	0.0586	0.0555	FFTW 1.06×
262K	0.2663	0.2779	RustFFT 1.04×
1M	1.5538	1.7654	RustFFT 1.14×
4M	8.0943	8.7243	RustFFT 1.08×

Across all tested sizes, RustFFT and FFTW differ by at most 14%. FFTW is marginally faster at $N \leq 65$K (6–12%), while RustFFT is marginally faster at $N \geq 262$K (4–14%). The two libraries are in the same performance class on this platform. Because SparseDSP's largest measured speedups occur at $N \geq 262$K—precisely the regime where RustFFT matches or exceeds FFTW—the reported speedups in the large-N, low-k regimes emphasized in this evaluation are robust to dense-library choice. At smaller N, where FFTW is slightly faster, the reported SparseDSP speedups should be interpreted as modestly optimistic (within the 6–12% calibration gap). This calibration compares dense transform kernels only; the benchmark-reported Dense times in subsequent sections include the full Dense post-processing pipeline (FFT + top-k selection). The calibration therefore validates FFT-kernel competitiveness, not end-to-end detector equivalence. A separate off-grid and real-payload dense-backend comparison confirmed similar near-parity under Path A and Path C conditions.

III. DOMAIN REQUIREMENTS

A. Domain Parameter Mapping

The benchmark framework uses abstract parameters: signal length N, sparsity k, and signal characteristics. Table VI maps these parameters to physical quantities across defense, commercial, and scientific sensing domains.

B. FMCW Radar Waveform Characteristics

In FMCW radar, the transmitter emits a linearly swept chirp with bandwidth B over duration T_{chirp} [1], [9]. The received signal from a target at range R produces a beat frequency:

$$f_{\text{beat}} = \frac{2BR}{cT_{\text{chirp}}} \tag{1}$$

where c is the speed of light. Range resolution is $\Delta R = c/(2B)$. Each resolvable target at a distinct range maps to a distinct frequency bin in the beat spectrum, producing an inherently sparse frequency-domain representation.

77 GHz automotive: $B = 4$ GHz, $T_{\text{chirp}} \approx 50$ μs, $f_s \approx 20$ MHz, yielding $N \approx 1$K beat-signal samples per chirp, $\Delta R = 3.75$ cm [1]. A coherent data cube (e.g., 1024

TABLE VI
BENCHMARK PARAMETER MAPPING ACROSS SENSING DOMAINS

Parameter	FMCW Radar	Airborne AESA	Military Sonar	SAR	ESM/ELINT
N	Chirp samples	CPI samples	CPI samples	Aperture samples	Survey window
k	Range targets	Air targets	Acoustic targets	Point scatterers	Active emitters
Freq bin	Beat freq $\rightarrow$ range	Doppler $\rightarrow$ velocity	Doppler $\rightarrow$ velocity	Azimuth freq	Emitter freq
Spread	Highway traffic	Dispersed aircraft	Open water	Distributed scene	Spread emitters
Band	Convoy cluster	Formation	Mine field	Urban cluster	Co-located radars

samples $\times$ 256 chirps $\times$ 4 Rx antennas) aggregates to $\sim$ 1M total samples per frame. Typical $k = 10$–50 targets per chirp (vehicles, pedestrians, infrastructure).

S-band weather radar (e.g., NEXRAD WSR-88D at $\sim$2.8 GHz): processes up to 1840 range gates per radial at 250 m resolution [21], with $k = 20$–50 weather returns per beam position.

C. Airborne and Naval Radar

Modern fighter AESA radars operate in multiple interleaved modes: air-to-air search, track-while-scan, ground mapping, and electronic attack [2]. Wideband pulse-compression modes produce $N = 262$K–1M range samples per CPI with $k = 10$–30 targets in a typical air-to-air engagement. The Doppler processing dimension (slow-time FFT across chirps) typically operates at $N_{\text{slow}} = 64$–256, below the sparse crossover; the fast-time range compression at $N = 262$K+ is the target for sparse engines.

Naval phased array radars (AN/SPY-6 for Aegis, SAMPSON for Type 45) perform simultaneous horizon search and volume surveillance [3], [22]. With $N = 131$K–524K samples per beam dwell and $k = 20$–100 contacts (surface vessels, aircraft, missiles), the operating point maps directly to the regime where SparseDSP achieves measured speedups of 4–5$\times$ at low sparsity in the evaluated x86 setup.

Ballistic missile defense radars (AN/TPY-2, THAAD) use wideband discrimination modes at $N = 524$K–2M for target classification, with $k = 5$–20 objects per track corridor. Passive radar systems exploiting FM/DVB-T/LTE illuminators of opportunity compute cross-ambiguity functions via FFT over long coherent intervals ($N = 131$K–2M) with sparse range-Doppler outputs ($k = 5$–20 aircraft or vehicles) [23]. Inverse SAR (ISAR) imaging of ships and aircraft uses cross-range FFTs over long coherent dwell times ($N = 262$K–1M) to resolve sparse dominant scatterers ($k = 10$–50) [24].

D. Military and Commercial Sonar

Mid-frequency active sonar operates at 1–10 kHz bandwidth with CPI of 5–60 seconds at $f_s = 25$ kHz [4], [5]:

Submarine hull-mounted: $N = 262$K–1.5M samples per CPI, $k = 5$–20 targets (submarines, surface vessels, bottom features). Onboard processing is constrained by platform SWaP-C (size, weight, power, and cost) limits, motivating efficient long-CPI detection algorithms.

Towed array: Extended CPIs of 30–60 seconds at $f_s = 25$ kHz yield $N = 750$K–1.5M samples. Long-range passive detection with $k = 5$–15 narrowband tonals from distant contacts.

ASW sonobuoy: $N = 131$K–262K per CPI, $k = 5$–10 targets. Air-deployed sonobuoys (e.g., AN/SSQ-53 family) operate on seawater-activated batteries with RF transmitter power of approximately 1 W nominal, imposing tight computational efficiency constraints.

Synthetic aperture sonar (SAS) for mine countermeasures and seabed imaging uses FFT-based azimuth processing analogous to SAR, with $N = 1$M–2M and sparse highlight scatterers ($k = 10$–100) [25].

E. Synthetic Aperture Radar

SAR processing is inherently two-dimensional [7]: range compression operates on each range line ($N = 4$K–8K for typical spaceborne SAR), while azimuth compression operates on the full synthetic aperture ($N = 1$M–4M).

Spaceborne SAR (military reconnaissance, Earth observation): Full-aperture azimuth at $N = 1$M–4M with $k = 50$–100 point scatterers. SmallSat/CubeSat SAR onboard processing is typically constrained to the order of 10–50 W, motivating sublinear-complexity algorithms for azimuth compression.

Airborne SAR: Similar N range with tighter latency requirements for real-time tactical imagery.

F. Electronic Warfare and Spectrum Surveillance

ESM/ELINT intercept receivers perform wideband spectral surveillance to detect, classify, and geolocate radar emitters [6], [26]. These systems cover frequency ranges such as 2–18 GHz; instantaneous digitized bandwidth is architecture-dependent, but at 1–2 GS/s digitization over 0.5–2 ms capture windows, each survey produces $N = 0.5$M–4M samples. The emitter environment is inherently sparse: $k = 10$–50 active emitters in a surveillance band, with the remainder of the spectrum containing only noise.

Low probability of intercept (LPI) radar detection requires high-sensitivity spectral analysis of weak FMCW, polyphase, and noise-like waveforms [6]. SparseDSP dispatches these large-N, low-k configurations to an internally selected exact engine, achieving $> 19\times$ speedup over Dense at $N \geq 2$M.

Space situational awareness radars for orbital debris tracking operate at similar N scales with $k = 10$–100 debris objects per beam dwell.

SparseTech Technical Papers

G. FMCW LiDAR

FMCW LiDAR at 1550 nm (eye-safe) uses coherent detection with sweep bandwidths up to 200 MHz [8]. Per-pixel sample counts depend on chirp duration and ADC rate (e.g., a 200 μs chirp at 10 MS/s yields $N \approx 2$K samples per pixel). Each scene reflector at a distinct range maps to a beat frequency, with $k = 50$–600 reflectors depending on scene complexity.

H. Detection Timing Across Domains

Table VII maps the benchmark timing data to defense and commercial sensing operating points ($k = 10$, spread placement, averaged across conditions):

TABLE VII
DETECTION TIMING: SPARSEDSP VS DENSE FFT ($k = 10$, SPREAD)

N	**Dense** (ms)	**SparseDSP** (ms)	**Speedup**	**Domain**
16K	0.257	0.065	4.05×	Short-CPI sonar
65K	0.903	0.211	4.34×	FMCW radar
262K	4.792	1.173	4.18×	AESA / naval
524K	10.013	2.337	4.87×	BMD / LiDAR
1M	22.103	3.630	8.91×	Wideband FMCW
2M	46.616	4.385	19.45×	SAR / ESM
4M	109.802	8.514	38.38×	SAR azimuth / EW

At the larger-N FMCW, AESA, and sonar scales considered here ($N \geq 65$K), the evaluated x86 benchmarks show 4–38× speedup at $k = 10$ relative to the Dense baseline. The speedup grows with N because Dense scales as $O(N \log N)$ while SparseDSP's internal engines scale sublinearly in N.

I. Regime Mapping

SparseDSP's relative performance is regime-dependent:

- **Legacy radar ($N < 32$K)**: Dense preferred. Sparse engine overhead exceeds FFT cost. Includes pulse-Doppler, short-range SAR range lines, and legacy naval search radar.
- **Modern FMCW, AESA, and sonar ($N = 65$K–262K)**: SparseDSP achieves 4–5× speedup at low sparsity ($k \leq 30$). Covers automotive FMCW, fighter AESA, naval phased array, and submarine sonar CPI.
- **Wideband FMCW, SAR, and EW ($N \geq 1$M)**: SparseDSP achieves 9–38× speedup. Covers full-aperture SAR, ESM survey windows, towed-array sonar, and BMD discrimination.
- **High sparsity ($k = 150$–200)**: SparseDSP is the only engine achieving exact recovery. Dense top-k fails to produce correct output at these sparsity levels (avg recall 0.7528, exact rate 0/960). SparseDSP incurs higher wall-clock cost (median 497.9 ms vs Dense 24.1 ms) but is the correct choice for exact reconstruction.

Explicit non-claim: This work does not propose replacing legacy pulse-Doppler radar processing with sparse engines. Dense FFT is the correct choice at pulse-Doppler scales and low SNR. At high sparsity ($k = 150$–200), SparseDSP achieves exact recovery where Dense top-k does not, but at higher wall-clock cost; for applications where approximate top-k detection is acceptable, Dense remains faster. The sparse regime is most relevant to larger-N, low-to-moderate-k wideband configurations.

IV. EXPERIMENTAL METHODOLOGY

A. Hardware, Software, and Benchmark Fairness

Benchmarks execute on a single-socket Intel Core i9-14900KF (24 cores / 32 threads) with 192 GiB DDR5 and AVX2 SIMD support, running Windows 11 (build 26200). All tests are single-threaded; GPU (RTX 5070 Ti) is not used. Rust 1.75+ compiler with release optimizations and LTO enabled, FP32 precision. Absolute timings are within-machine relative results.

Benchmark configuration: Tests run on a single P-core with thread affinity pinned (core 0), Intel Turbo Boost enabled (default behavior), Windows High Performance power plan, and process priority set to High. Each engine is warmed with one untimed invocation per (N, k) configuration to ensure FFT plan creation, buffer allocation, and precomputation occur outside the timed region. The timed region covers only the core algorithm execution (frequency discovery and bin output), not input generation or result validation. Memory allocations within the timed region are part of the algorithm cost.

Benchmark fairness checklist:

- Same hardware platform for all engines and Dense baseline
- Same FP32 precision throughout
- Same single-threaded execution, same core affinity
- Same warmup protocol (one untimed run per configuration)
- Same timing region (core algorithm only; memory allocations included)
- Plan reuse and buffer reuse applied to Dense baseline
- Black-box optimization barriers preventing dead code elimination
- All dispatch parameters fixed before benchmark execution

Hardware proxy limitation: This is a desktop-class x86 CPU, not the DSP/FPGA/ASIC hardware used in production radar, sonar, EW, or LiDAR systems (e.g., Xilinx RFSoC, TI TDA4x, NXP S32R, BAE Systems radar processors, or spaceborne radiation-hardened FPGAs). All timing results characterize algorithmic complexity ratios between SparseDSP and Dense on a common platform. Crossover boundaries and absolute timings will shift on embedded or accelerator platforms; however, the relative ordering may remain similar in some regimes, although this requires platform-specific verification because they reflect algorithmic complexity differences ($O(N \log N)$ vs sublinear-N). Constant-factor shifts from vector width, cache hierarchy, fixed-point arithmetic, or vendor-optimized FFT libraries (FFTW, MKL, cuFFT) could move crossover boundaries, particularly in close regimes. Additionally, FFTs at $N \geq 1$M are heavily memory-bandwidth-bound; sparse engines often trade sequential memory access

SparseTech Technical Papers

SparseTech Technical Papers

for irregular patterns (modular indexing, non-contiguous bin reads), so crossover boundaries on embedded platforms will depend on cache hierarchy and memory bandwidth, not solely on arithmetic throughput. Re-measurement on each target platform is required.

B. Dense FFT Baseline Configuration

Library: RustFFT 6.4.1 pure Rust implementation [20]. Optimizations applied to ensure fair comparison:

- Plan reuse: FFT plan created once, reused across iterations
- Buffer reuse: Complex signal buffer pre-allocated
- Efficient top-k: Partial selection via select-nth-unstable-by avoiding full sort
- Black-box optimization barriers preventing dead code elimination

C. Path A: Synthetic Benchmark

Path A validates SparseDSP across 4992 synthetic configurations:

Low-k suite (4032 configurations): Signal lengths $N =$ 16K–4M, sparsity $k =$ 2–30, spread and band placement patterns. Each configuration tests SparseDSP against Dense FFT on synthetic sparse signals with known ground truth.

High-k suite (960 configurations): Same N range, sparsity $k =$ 100–200, where sparse engines face increasing overhead.

Exactness definition (Path A): SparseDSP output is *exact* if the set of detected frequency bins matches the Dense FFT reference output (support match) and detected amplitudes agree within floating-point tolerance ($< 10^{-5}$ relative error).

System-level reporting: For each configuration, SparseDSP receives only the fixed-length input block, estimates sparsity internally, dispatches to its internal engines, and reports the fastest exact result. No oracle sparsity label is supplied. Results reflect aggregate system-level speedup (SparseDSP wall-clock time divided by Dense FFT wall-clock time).

D. Path B: Impairment Robustness Benchmark

Path B validates SparseDSP under signal impairments that degrade idealized synthetic conditions:

- **Off-grid frequencies**: Tones placed at non-integer bin positions
- **Fading channels**: Rayleigh, Rician ($\kappa = 3$ dB and $\kappa =$ 10 dB) amplitude fluctuations
- **Clutter-like interference**: K-distributed and Weibull-shaped ($\beta = 12$ dB) clutter models
- **AWGN**: Additive white Gaussian noise at varying SNR

Exactness definition (Path B): Same as Path A (support match and amplitude match within tolerance), applied after signal propagation through the impairment model. For off-grid tones, "support match" means matching the *dominant DFT bin* (the bin receiving the largest share of the tone's energy after spectral leakage), consistent with the Dense FFT reference. Both systems identify the same dominant-bin set; the comparison is whether algebraic reconstruction recovers the same dominant bins as magnitude-based selection under impairment.

Both SparseDSP and Dense FFT (with top-k heuristic) are evaluated under identical impairment conditions. Path B is designed to test the resilience of *algebraic reconstruction* (CRT-based bin identification, structured gating) versus *magnitude-based bin selection* (top-k ranking) under realistic signal degradation. A production Dense pipeline would typically employ CFAR or matched filtering rather than top-k; Path B isolates the base resilience of the bin-identification step itself.

E. Path C: Curated Real-Payload Spectral Fidelity

Path C validates SparseDSP on curated subsets from six public datasets:

- **nexrad**: NOAA weather radar returns [21]
- **oxford_radar**: Oxford RobotCar automotive radar [23]
- **argoverse2**: Argoverse 2 autonomous driving LiDAR/radar
- **deepmimo**: DeepMIMO channel model [27]
- **nuscenes**: nuScenes sensor fusion dataset
- **argos**: Argos satellite telemetry

Each dataset is evaluated on three tasks: detection (identify occupied frequency bins), reconstruction (recover amplitudes), and clutter suppression (remove non-target spectral components). All tasks compare SparseDSP output against Dense FFT reference output.

Exactness definition (Path C): Detected frequency bins exactly match the Dense FFT reference output. All tasks achieved exact recovery (1.0) across all six datasets.

Sparsity estimation: All Path C experiments evaluated SparseDSP with its internal sparsity estimation enabled. Sparsity k was *not* supplied as an external oracle parameter; the system estimated the effective sparsity from each input block at runtime before recovery. The exact results reported above therefore include the effect of the system's own sparsity-estimation stage. Typical ranges across the six datasets: $N =$ 16K–1M, $k =$ 4–50, with the majority of records falling in the low-k regime ($k < 30$). The datasets are curated subsets from public sources chosen to represent realistic spectral signatures, not raw unprocessed sensor data streams with arbitrary clutter or unknown scene composition.

Diagnostic metrics: Four metrics characterize system behavior per dataset. Let L denote the number of internal sub-transforms (views), m_ℓ the number of bins in view ℓ, S the recovered support with $|S| = k$, $c_\ell(f, g) = \mathbf{1}[h_\ell(f) = h_\ell(g)]$ the collision indicator for frequencies $f \neq g$ under view ℓ, and $r_\ell[b]$ the number of support elements mapped to bin b in view ℓ:

- **CM** (collision metric): The fraction of frequency pairs that collide across views, normalized by the maximum possible collisions:

$$\text{CM} = \frac{1}{L\binom{k}{2}} \sum_{\ell=1}^{L} \sum_{\substack{f,g \in S \\ f<g}} c_\ell(f, g) \tag{2}$$

SparseTech Technical Papers

CM $= 0$ indicates no collisions (every pair separated in every view); CM $\to 1$ indicates pervasive aliasing. Values below 0.15 indicate successful reconstruction in the benchmark corpus.

- **U** (residue uniformity): The normalized chi-squared statistic measuring deviation of the bin-occupancy distribution from the uniform ideal:

$$\mathrm{U} = \frac{1}{L}\sum_{\ell=1}^{L}\frac{m_\ell}{k}\sum_{b=0}^{m_\ell-1}\left(r_\ell[b]-\frac{k}{m_\ell}\right)^2 \tag{3}$$

  U $= 0$ indicates perfectly uniform distribution; larger values indicate clustering. Values below 2.0 indicate successful runs in the evaluated corpus.
- **SAT** (saturation): The ratio of peak bin occupancy to the average occupancy, averaged across views:

$$\mathrm{SAT} = \frac{1}{L}\sum_{\ell=1}^{L}\frac{\max_b\, r_\ell[b]}{k/m_\ell} \tag{4}$$

  SAT $= 1$ indicates no bin exceeds the average load; higher values indicate hot-spot bins. Values below 5.0 indicate successful singleton isolation in the evaluated system.
- **API** (aggregate performance indicator): The wall-clock speedup ratio:

$$\mathrm{API} = \frac{t_{\text{Dense}}}{t_{\text{SparseDSP}}} \tag{5}$$

  API > 1 indicates SparseDSP completed the task faster than Dense; API < 1 indicates Dense completed faster. In both cases, SparseDSP achieves exact recovery (1.0) — the API ratio reflects task-specific timing variation at the operating point's (N, k), not a correctness difference.

The thresholds reported above (CM < 0.15, U < 2.0, SAT < 5.0) are empirical internal acceptance criteria derived from successful reconstruction regions in the benchmark corpus, not universal theoretical bounds. Because CM, U, and SAT depend on internal view construction parameters (L, m_ℓ, h_ℓ), they serve as system diagnostics characterizing reconstruction health.

F. Definition of Exactness

Across all three paths, "exact" has a rigorous operational definition:

1) **Support match**: The set of frequency bin indices returned by SparseDSP equals the set returned by Dense FFT reference.
2) **Amplitude match**: For each matched bin, $|A_{\text{sparse}} - A_{\text{dense}}|/|A_{\text{dense}}| < 10^{-5}$.
3) **Determinism**: Repeated invocations on identical input produce identical output.

Exactness here means equivalence to the Dense DFT-plus-selection reference output, not physical-target ground-truth detection optimality. This definition differs from the tolerance-based recall used in prior work [10], [11] and provides a binary pass/fail criterion aligned with the determinism expectations of safety-critical processing chains.

G. System-Level Reproducibility Envelope

This paper specifies the evaluated system-level benchmark conditions, consisting of: (1) benchmark paths and exactness criteria (Sections IV-C–IV-E); (2) tested N and k regimes; (3) impairment families and off-grid interpretation rules; (4) Dense baseline configuration including FFTW calibration (Table V); (5) hardware, software, and execution conditions; and (6) observable outputs in the form of exactness, timing, and regime-dependent behavior. The internal engine-selection mechanism and per-engine implementation details are not part of this envelope. The evaluation protocol and the expected form of system-level behavior under that protocol are fully specified.

V. Application Context

The following application examples are deployment interpretations derived from the measured Path A/B/C evidence and the literature-grounded parameter mappings. They are not validated end-to-end studies of radar, sonar, EW, SAR, or LiDAR receiver chains.

A. Sparse Target Detection in FMCW Radar

The beat-frequency spectrum of an FMCW radar chirp is inherently sparse: each resolvable target at range R maps to a single beat frequency $f_{\text{beat}} = 2BR/(cT_{\text{chirp}})$ [9]. For wideband FMCW range processing or aggregated fast-time workloads with effective transform length $N = 262$K and $k = 10$ targets, SparseDSP processes this operating point in 1.173 ms versus Dense at 4.792 ms (Table VII). Note that standard automotive per-chirp beat-signal lengths are $\sim$1K samples (Section II); the $N = 262$K operating point represents wideband or aggregated configurations, not a single standard chirp.

FMCW radar presents a favorable application for sparse FFT because: (1) each target produces a single dominant beat frequency (no multipath spreading in the beat domain), (2) the sparsity level k is bounded by the physical scene, and (3) the chirp repetition rate (up to 10 kHz for fast-chirp FMCW) creates per-chirp latency budgets that can make Dense FFT computationally expensive at large N.

For range-Doppler processing, a second FFT across the slow-time dimension (chirp-to-chirp) resolves velocity. This slow-time FFT typically operates at $N_{\text{slow}} = 64$–256 (number of chirps per frame), below the sparse crossover. SparseDSP dispatches the slow-time FFT to Dense while dispatching the fast-time (beat-frequency) FFT to an internally selected sparse engine.

B. Airborne AESA and Naval Phased Array Radar

Fighter AESA radars in air-to-air search mode process wideband pulse-compressed returns with $N = 262$K–1M range samples and $k = 10$–30 airborne targets per scan volume [2], [28]. At $N = 262$K, $k = 10$: SparseDSP at 1.173 ms provides 4.18$\times$ speedup over Dense, indicating lower transform-stage compute cost in the evaluated benchmark.

Naval surface radar tracking $k = 20$–100 contacts at $N = 131$K–524K maps to the regime where SparseDSP

provides speedup at low-to-moderate k. For missile defense discrimination at $N = 524$K–2M, the 5–19× sparse speedup reduces transform-stage runtime in the benchmark; effects on end-to-end compute margin would require target-platform evaluation across multiple threat corridors.

C. Military and ASW Sonar

Submarine sonar CPI processing at $N = 262$K–1.5M with $k = 5$–20 targets places the operating point where SparseDSP shows measured speedup in the evaluated x86 benchmark. At $N = 524$K, $k = 10$: SparseDSP at 2.337 ms versus Dense at 10.013 ms (4.87× speedup).

ASW sonobuoy deployments face tight power constraints (< 2 W total system power). On the x86 benchmark platform (i9-14900KF), SparseDSP achieves a 4–5× complexity reduction at $N = 262$K, $k = 10$. Whether this complexity ratio translates to meaningful power savings on embedded sonobuoy processors depends on the target architecture and is not validated here.

Towed-array processing at $N = 750$K–1.5M with $k = 5$–15 narrowband tonals achieves 9–19× speedup on the x86 benchmark platform (Table VII). The asymptotic complexity difference is architecture-independent, although observed runtime ratios are platform-dependent; absolute timing on submarine processing systems is not measured.

D. Worked Example: Submarine Sonar CPI Processing

Consider a submarine hull-mounted sonar processing a coherent processing interval at mid-frequency active sonar:

- Sample rate: $f_s = 25$ kHz
- CPI duration: 10 seconds
- Signal length: $N = 250$K samples
- Dominant targets: $k = 5$ (submarines, surface vessels, bottom features)

Dense FFT processing:

- Per-CPI cost: $5 \times 250\text{K} \times 18 \approx 22.5$M FLOPs
- All 250K frequency bins produced and passed to narrowband detector
- On x86 (i9-14900KF): ≈4.5 ms per CPI (interpolated from Table VII)

SparseDSP processing:

- Per-CPI cost: sublinear in N; $k = 5$ dominant tonals recovered directly
- Only 5 frequency bins passed to downstream tracker
- On x86: ≈1.1 ms per CPI (interpolated from measured speedup ratios)
- Speedup: ≈4× on x86

Result: The 4× complexity reduction is measured on x86. Whether this translates to meaningful power or latency savings on submarine processing hardware depends on the target architecture. For platforms where the FFT stage is a bottleneck (e.g., onboard processing constrained by SWaP-C), the complexity ratio indicates potential margin; for platforms where other stages dominate, the FFT reduction may have limited system-level impact.

This example uses small k where SparseDSP's advantage is strongest. At higher k (e.g., $k = 100$ contacts in a dense shipping lane), Dense FFT is faster but does not produce exact output; SparseDSP achieves exact recovery at higher wall-clock cost (Section III-I).

E. SAR and Space-Based Radar

SAR range-line processing at $N = 4$K–8K with $k \sim 200$ scatterers falls in the Dense regime (below crossover, high k/N ratio). Full-aperture azimuth processing at $N = 1$M–4M with $k = 50$–100 point scatterers falls within the large-N, moderate-k regime where speedups were observed in the benchmark [7].

The 2D separable structure means each dimension is dispatched independently, with Dense handling range lines and SparseDSP handling azimuth compression for sparse scenes. For spaceborne SAR constellations, the 9–38× complexity reduction at $N \geq 1$M reduces transform-stage computational cost in the benchmark; relevance to onboard power constraints would require target-platform measurement.

F. Electronic Warfare Spectrum Surveillance

ESM/ELINT intercept receivers process $N = 1$M–4M samples per survey window with $k = 10$–50 active emitters [6], [26]. On the x86 benchmark platform: at $N = 2$M, $k = 10$, SparseDSP at 4.385 ms versus Dense at 46.616 ms (19.45×); at $N = 4$M, SparseDSP at 8.514 ms versus Dense at 109.802 ms (38.38×). All timing is x86-relative (Table VII).

For LPI radar detection, the combination of large N (to achieve sufficient frequency resolution for detecting weak FMCW chirps) and small k (few emitters present) places the operating point in the regime where SparseDSP corresponds to one of the largest measured speedup regimes in this evaluation.

VI. Validated Performance Results

Table VIII summarizes the scope and coverage of the three validation paths.

TABLE VIII
BENCHMARK ACCOUNTING SUMMARY

Path	Configs	N range	k range	Conditions
A low-k	4032	16K–4M	2–30	4 placements × 24 channel/SNR/offset
A high-k	960	262K–4M	150–200	4 placements × 24 conditions
B	2160	16K–1M	16–100	6 channels, off-grid, fading
C	102	16K–1M	4–50	6 datasets × 3 tasks

SparseTech Technical Papers

SparseTech Technical Papers

A. Path A System-Level Results

SparseDSP achieves 100% exact recovery across all 4992 Path A configurations (4032 low-k + 960 high-k). **Low-k speedup distribution** (4032 configurations, $k = 2$–30):

- Median speedup: 3.93×
- 10th percentile (P10): 3.03×
- 90th percentile (P90): 24.75×

High-k correctness regime (960 configurations, $k = 150$–200):

- SparseDSP achieves 100% exact recovery (960/960)
- Dense top-k achieves 0% exact recovery at these operating points (avg recall 0.7528)
- SparseDSP incurs higher wall-clock cost (median 497.9 ms vs Dense 24.1 ms) but produces correct output where Dense does not

This correctness regime reflects a fundamental limitation of magnitude-based bin selection: at high k, many bins have similar energy levels, and top-k ranking frequently selects incorrect bins. SparseDSP's algebraic reconstruction (CRT-based identification) is not affected by amplitude distribution and maintains exact recovery. The tradeoff is correctness for latency, not a performance "loss." In production deployment, the dispatch function can prefer Dense FFT at high k when approximate top-k detection is acceptable; the benchmark reports both regimes for completeness.

Speedup scaling with N: Table IX presents system-level speedup at $k = 10$ across the tested signal-length range.

TABLE IX
SPARSEDSP SPEEDUP VS DENSE FFT ($k = 10$, SPREAD PLACEMENT)

N	Dense (ms)	SparseDSP (ms)	Speedup
16K	0.257	0.065	4.05×
65K	0.903	0.211	4.34×
262K	4.792	1.173	4.18×
524K	10.013	2.337	4.87×
1M	22.103	3.630	8.91×
2M	46.616	4.385	19.45×
4M	109.802	8.514	38.38×

SparseDSP speedup grows with N because Dense FFT scales as $O(N \log N)$ while SparseDSP's internal engines scale sublinearly in N. The 38.38× speedup at $N = 4$M illustrates the measured speedup at this tested operating point. Note that the 4.05× speedup at $N = 16$K reflects benchmark-lane behavior at $k = 10$; production dispatch policies may conservatively prefer Dense at small N as a practical safety margin, since sparse engine overhead is closer to FFT cost in this regime.

B. Path B System-Level Impairment Robustness

Table X presents system-level exactness under six channel impairment models:

SparseDSP achieves ≥99.44% exact recovery across all six channel models. Across 2160 total impaired trials (360 per channel model), the remaining failures ($\leq$ 12 cases) concentrate in Rician-κ=3 dB fading at high off-grid offsets (0.5 bin), where deep fading occasionally pushes a target tone below the reconstruction noise floor; these cases would be candidates for the Dense fallback path in the evaluated design. Dense FFT (with top-k heuristic) achieves 0.00–0.28% exact recovery under the same conditions. The asymmetry arises because SparseDSP's internal engines use algebraic reconstruction (CRT-based identification, structured gating) rather than magnitude-based bin selection; impairments that redistribute energy across bins defeat top-k but often do not prevent algebraic recovery under the tested conditions.

TABLE X
PATH B BIN-IDENTIFICATION EXACTNESS: ALGEBRAIC RECOVERY VS DENSE TOP-k HEURISTIC

Channel Model	SparseDSP	Dense Top-k
AWGN	≥99.44%	0.28%
K-distributed clutter	≥99.44%	0.00%
Rayleigh fading	≥99.44%	0.00%
Rician ($\kappa = 3$ dB)	≥99.44%	0.00%
Rician ($\kappa = 10$ dB)	≥99.44%	0.00%
Weibull ($\beta = 12$ dB)	≥99.44%	0.00%

Dense top-k exact rate reflects the heuristic's inability to identify correct bins under impairment; the FFT itself is exact. SparseDSP lane acceptance: both sparse lanes PASS with min recall/precision = 1.0.

Interpretation: The 0.00% Dense result does not indicate that Dense FFT is broken. The Dense FFT transform is exact. The top-k *post-processing heuristic* fails because channel impairments spread signal energy beyond the k strongest bins. A matched filter or CFAR detector on Dense FFT output would perform differently. The comparison isolates the bin-identification quality of SparseDSP's algebraic approach versus Dense's magnitude-ranking approach.

C. Path C Real-Payload Results

Table XI presents SparseDSP timing and exactness on six public datasets:

All six datasets achieve exact recovery (1.0) across all three tasks. Each task operates at a fixed (N, k) operating point: detection at $N = 65$K, $k = 25$; reconstruction at $N = 262$K, $k = 51$; clutter suppression at $N = 1$M, $k = 102$. Detection timing ranges from 0.25 ms (deepmimo) to 1.99 ms (argos). Reconstruction timing is 4.39–6.46 ms. Clutter suppression timing varies from 11.43 ms (deepmimo) to 49.39 ms (nexrad), reflecting dataset-dependent signal structure at the same (N, k) operating point.

Dispatch reliability: Across all 36 Path C sparse-engine evaluations (6 datasets × 3 tasks × sparse engine), SparseDSP exhibited a 0% fallback rate and zero dispatch or runtime errors. Certified and raw exact rates were 1.0000 for all sparse rows, indicating that the reported Path C results reflect stable end-to-end sparse execution rather than recovery through Dense fallback.

Diagnostic metrics per dataset are presented in Table XII:

Collision metrics are low across all datasets (CM $\leq$ 0.130), consistent with low measured bin interference during reconstruction. SparseDSP achieves exact recovery (1.0) on all six

TABLE XI
PATH C REAL-PAYLOAD RESULTS (ALL TASKS: EXACT = 1.0)

Dataset	Detection (ms)	Reconstruction (ms)	Clutter (ms)	Exact	Domain Connection
argos	1.99	6.18	13.26	1.0	Satellite telemetry
argoverse2	0.27	5.28	11.55	1.0	Autonomous driving
deepmimo	0.25	4.39	11.43	1.0	MIMO channel
nexrad	0.28	6.13	49.39	1.0	Weather radar
nuscenes	0.41	6.46	44.37	1.0	Sensor fusion
oxford_radar	0.48	6.18	14.24	1.0	Automotive radar

TABLE XII
PATH C DIAGNOSTIC METRICS

Dataset	CM	U	SAT	API
argos	0.130	1.93	4.61	4.01
argoverse2	0.004	0.49	1.24	0.49
deepmimo	0.006	0.56	1.38	0.55
nexrad	0.008	0.72	1.87	0.88
nuscenes	0.012	0.83	2.14	1.12
oxford_radar	0.009	0.61	1.56	0.72

CM = collision metric, U = residue uniformity, SAT = saturation, API = dense-to-sparse performance ratio.

datasets and all three tasks — the API ratio reflects only task-specific timing variation at each dataset's (N, k) operating point. API > 1 (argos, nuscenes) indicates SparseDSP completed the task faster than Dense; API < 1 (argoverse2, deepmimo) indicates Dense completed faster for that task. These are timing differences, not correctness differences: SparseDSP produces exact output in every case.

D. Regime Applicability Summary

Table XIII maps measured performance to representative application contexts:

TABLE XIII
SPARSEDSP REGIME APPLICABILITY

N Range	Speedup	Application
$\leq$ 16K	k-dependent†	Pulse-Doppler, SAR line
16–65K	4.0–4.3×	Sonar CPI, short FMCW
65–524K	4.2–4.9×	FMCW, AESA, Sonar
524K–1M	4.9–8.9×	BMD, wideband FMCW
1–2M	8.9–19.5×	SAR, ESM, Weather
2–4M	19.5–38.4×	SAR azimuth, EW
High-k correctness regime:		
Any N, $k >$ 100	Exact via SparseDSP only	Dense top-k inexact

†SparseDSP can win at low k even at small N; production dispatch may conservatively prefer Dense in this regime.

VII. PRODUCTION DEPLOYMENT CONSIDERATIONS

A. Dispatch Policy

SparseDSP selects an engine according to its fixed dispatch policy based on signal parameters. The dispatch function is deterministic: identical inputs produce identical engine selection. The evaluated dispatch behavior has the following observed characteristics:

- **Scale awareness**: Below empirical crossover ($N < 32$K), Dense is selected. Sparse engine overhead exceeds FFT cost at legacy radar scales.
- **Sparsity awareness**: At high k relative to N, Dense is preferred. SparseDSP does not force sparse processing when Dense is faster.
- **Fallback guarantee**: If the dispatched sparse engine fails post-detection verification, the system falls back to Dense FFT. In the evaluated design, the fallback path trades a failed sparse attempt for additional latency rather than silent acceptance of that result.

All dispatch parameters were fixed before benchmark execution. The observable system-level timing and exactness constitute the specification for the dispatch logic.

B. Defense System Requirement Context

Processing budgets vary by domain: fast-chirp automotive FMCW radars commonly use chirp durations of 20–100 μs [1]; multifunction AESA radars allocate beam dwells from sub-millisecond to several milliseconds depending on task [2], [22]; SmallSat SAR onboard processing is typically constrained to 10–50 W. On the x86 benchmark platform (i9-14900KF, single-thread), SparseDSP at $N = 262$K achieves 1.173 ms (4.18× reduction versus Dense). Whether this complexity reduction translates to meeting production budgets depends on the target hardware architecture (DSP/FPGA/ASIC); this report establishes the algorithmic complexity ratio, not a hardware-specific deployment claim.

C. Workload Budget Analysis

To illustrate the benchmark-scale implications of the complexity difference, we present concrete workloads showing the Dense FFT computational burden versus SparseDSP. All results characterize relative runtime behavior on x86; absolute timings will differ on production hardware, but complexity differences are expected to persist across architectures, though exact ratios are platform-dependent.

SparseTech Technical Papers

Wideband FMCW radar: $N = 262$K (aggregated or wideband configuration), $k = 10$. On x86 (i9-14900KF, single-thread): Dense 4.792 ms, SparseDSP 1.173 ms (4.18× reduction). Standard automotive per-chirp lengths (∼1K samples) fall below the sparse crossover; this operating point represents wideband or aggregated processing workloads. Absolute timing on production radar processors is not measured; only the complexity ratio is reported.

Fighter AESA radar: $N = 262$K, $k = 20$, < 1 ms dwell budget. On x86: Dense 4.792 ms, SparseDSP 1.173 ms (4.18× reduction). Both exceed the 1 ms budget on this platform; the complexity ratio is reported, not absolute embedded timing.

Naval phased array: $N = 524$K, $k = 30$, simultaneous track and search. On x86: Dense 10.013 ms, SparseDSP 2.337 ms (4.87× reduction). The complexity ratio indicates reduced transform-stage runtime in the benchmark; system-level compute margin would require target-platform analysis.

Spaceborne SAR: $N = 4$M azimuth, $k = 50$, < 50 W budget. On x86: Dense 109.802 ms, SparseDSP 8.514 ms (38.38× reduction). Absolute timing on radiation-hardened spaceborne processors is not measured; the complexity ratio is reported.

ESM/ELINT survey: $N = 2$M, $k = 10$–50 emitters. On x86: Dense 46.616 ms, SparseDSP 4.385 ms (19.45× reduction). The complexity ratio is reported; absolute timing on EW digital receivers is not measured.

D. Production Caveats

- **Path C staged datasets**: Real-payload datasets are curated subsets at known sparsity, not raw sensor data. They validate spectral signature fidelity, not end-to-end pipeline integration with unknown sparsity estimation.
- **Dispatch scope**: The dispatch function is deterministic and validated at the system level.
- **High-k correctness regime**: At $k = 150$–200, SparseDSP achieves exact recovery but at higher wall-clock cost than Dense. Dense top-k does not produce exact output in this regime (avg recall 0.7528, exact rate 0/960). Production systems operating at high k should use SparseDSP when exact recovery is required, or Dense FFT when approximate top-k detection is acceptable.

E. System-Level ROI Translation

Benchmark speedup ratios translate to system-level resource implications as follows. All figures are analytical projections from measured x86 timing data, not direct platform measurements, and are labeled as deployment interpretations.

- **Processing margin**: A 4× speedup at $N = 262$K reduces the transform stage from 4.79 ms to 1.17 ms on x86, freeing 3.6 ms per dwell or chirp for other processing stages (beamforming, tracking, classification) under the same latency envelope. Platform-specific impact depends on the target architecture.
- **Beam revisit rate**: For multi-function AESA radar, the freed transform-stage compute could, in principle, support proportionally more beam dwells per scan period, subject to scheduling constraints and other processing bottlenecks.
- **Sonar CPI budget**: For submarine and ASW sonar, the complexity reduction indicates potential margin within onboard SWaP-C constraints. Whether this margin is operationally meaningful depends on the processing architecture and which pipeline stages dominate the compute budget.
- **SAR throughput**: At $N = 4$M (full-aperture azimuth), the 38× complexity reduction could enable processing proportionally more azimuth lines within a fixed power or time budget, subject to memory bandwidth and I/O constraints.

These projections illustrate how transform-stage complexity reductions could affect system-level resource allocation. They are deployment interpretations, not validated platform measurements.

VIII. Conclusion

This work validates SparseDSP, a regime-adaptive deterministic sparse FFT system, against Dense FFT across radar, sonar, electronic warfare, and LiDAR operating points.

In these experiments, SparseDSP estimates sparsity internally; in all reported experiments, including the six real-payload datasets in Path C, the system estimated sparsity from the input signal rather than receiving an oracle parameter. SparseDSP achieves 100% exact recovery across 4992 Path A configurations, ≥99.44% exact across six Path B channel impairment models (versus 0.00–0.28% for Dense top-k), and exact recovery (1.0) across six Path C real-payload datasets. At representative low-sparsity operating points ($k = 10$): 4.05× speedup at $N = 16$K, 4.87× at $N = 524$K, and 38.38× at $N = 4$M. At high support sizes ($k = 150$–200), SparseDSP achieves 100% exact recovery where Dense top-k does not (Dense exact rate 0%, avg recall 0.7528); SparseDSP incurs higher wall-clock cost but is the only option for exact reconstruction at these operating points.

Dense FFT is the correct choice at legacy pulse-Doppler scales ($N < 32$K) and when approximate top-k detection is acceptable. At high k (> 100), Dense is faster but does not produce exact output; SparseDSP is the correct choice when exact recovery is required. The evaluated results are most relevant to regimes where wideband sensors produce signals with $k \ll N$: FMCW radar, AESA pulse compression, naval phased array, submarine sonar CPI, full-aperture SAR azimuth, and wideband ESM survey windows.

All measurements are detection-only (frequency bin identification), pre-CFAR, and within-machine relative on x86. The principal output of this study is the measured system-level behavior under the stated protocol. A companion report validates the same framework for wireless spectrum sensing [19].

SparseTech Technical Papers

SparseTech Technical Papers

IX. Limitations

1) **Pre-CFAR only**: All results are detection-only (frequency bin identification). No amplitude recovery optimization, no CFAR detection threshold integration, no tracker-level performance evaluation. Algebraic sparse reconstruction can introduce structured residual artifacts (e.g., CRT aliasing patterns, incomplete peel residuals) that produce a non-flat noise floor differing from the white noise floor of a full Dense FFT. These structured artifacts could locally raise CFAR thresholds and reduce detection sensitivity in adjacent cells. The interaction between sparse reconstruction residuals and downstream CFAR processors (CA-CFAR, OS-CFAR) has not been evaluated.
2) **Hardware platform**: All results are from a desktop-class x86 CPU. No radar DSP (Xilinx RFSoC, TI TDA4x), sonar processor, EW digital receiver, space-borne FPGA, or LiDAR ASIC validation.
3) **Path C curated data**: Real-payload datasets are curated subsets chosen for realistic spectral signatures; they do not represent raw unprocessed sensor streams with arbitrary clutter or unknown scene composition.
4) **No external baselines**: This report benchmarks SparseDSP against the Dense FFT baseline. No comparison against external sparse FFT implementations (MIT sFFT, FFAST, etc.) or conventional radar sparse recovery methods (OMP, MUSIC, ESPRIT, compressed sensing) is included; such comparison is deferred to companion algorithm papers [16]–[18].
5) **Regime-dependent**: SparseDSP does not outperform Dense FFT in wall-clock time at all operating points. At high support sizes ($k = 150$–200), SparseDSP achieves exact recovery but at higher wall-clock cost than Dense. Dense top-k does not produce exact output in this regime (avg recall 0.7528, exact rate 0/960). The tradeoff is correctness for latency. At small N (< 32K), Dense is preferred. The dispatch function encodes these boundaries.
6) **Pulse-Doppler non-claim**: This work targets wideband FMCW radar, AESA modes, sonar CPI, SAR azimuth, and EW survey processing, not legacy pulse-Doppler FFT replacement.
7) **Timing relativity**: All timing comparisons are within-machine relative on a desktop-class x86 CPU (i9-14900KF). Crossover boundaries and absolute timings will shift on embedded, FPGA, or ASIC platforms; complexity differences may preserve similar trends, although this requires platform-specific measurement.
8) **Signal model**: Path A and Path B use synthetic sparse tones with controlled impairments. No evaluation on realistic combined radar clutter (ground, sea, weather), multipath, Doppler ambiguity, range-velocity coupling, sidelobe clutter, jamming, colored ocean noise, quantization, or phase noise simultaneously. Path C partially addresses this gap with real-world spectral signatures but at curated sparsity levels.
9) **Dispatch abstraction**: Validation assesses end-to-end system behavior, abstracting implementation-dependent engine selection boundaries.
10) **Sparsity estimation**: While SparseDSP estimates sparsity internally at runtime (and all reported results reflect this deployed behavior), this paper does not separately ablate the accuracy of the internal sparsity-estimation stage versus scene conditions or compare against alternative sparsity-selection heuristics.
11) **Stateless processing**: Because SparseDSP operates independently on each transform, this work does not exploit temporal tracking or cross-frame priors that could be evaluated in future work for slowly varying scenes.
12) **Benchmark scope**: 4992 Path A configurations, six Path B channel models, and six Path C datasets represent a broad but not exhaustive coverage of the (N, k) parameter space and real-world signal diversity.

Acknowledgment

The authors gratefully acknowledge the collaborative environment at SparseTech that made this research possible.

The theoretical and computational developments presented in this paper are part of the authors' broader research program on deterministic sparse FFT algorithms.

References

[1] S. Patole, M. Torlak, D. Wang, and M. Ali, "Automotive radars: A review of signal processing techniques," *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 22–35, Mar. 2017.
[2] G. W. Stimson, H. D. Griffiths, C. J. Baker, and D. Adamy, *Stimson's Introduction to Airborne Radar*, 3rd ed. Raleigh, NC, USA: SciTech Publishing, 2014.
[3] M. I. Skolnik, *Radar Handbook*, 3rd ed. New York, NY, USA: McGraw-Hill, 2008.
[4] D. A. Abraham, *Underwater Acoustics and Signal Processing*. Cham, Switzerland: Springer, 2019.
[5] R. J. Urick, *Principles of Underwater Sound*, 3rd ed. Los Altos, CA, USA: Peninsula Publishing, 1983.
[6] P. E. Pace, *Detecting and Classifying Low Probability of Intercept Radar*, 2nd ed. Norwood, MA, USA: Artech House, 2009.
[7] I. G. Cumming and F. H. Wong, *Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation*. Norwood, MA, USA: Artech House, 2005.
[8] B. Behroozpour, P. A. M. Sandborn, M. C. Wu, and B. E. Boser, "Lidar system architectures and circuits," *IEEE Communications Magazine*, vol. 55, no. 10, pp. 135–142, Oct. 2017.
[9] M. A. Richards, *Fundamentals of Radar Signal Processing*, 2nd ed. New York, NY, USA: McGraw-Hill Education, 2014.
[10] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Simple and practical algorithm for sparse fourier transform," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Kyoto, Japan, Jan. 2012, pp. 1183–1194.
[11] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Nearly optimal sparse fourier transform," in *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, New York, NY, USA, May 2012, pp. 563–578.
[12] ISO, "Iso 26262: Road vehicles – functional safety," 2018, 2nd edition.
[13] RTCA, "Do-178c: Software considerations in airborne systems and equipment certification," 2011.
[14] U.S. Department of Defense, "Mil-std-461g: Requirements for the control of electromagnetic interference characteristics of subsystems and equipment," 2015.

[15] X. Li and G. Caire, "A new class of deterministic sparse fft algorithms based on chinese remainder theorem," *IEEE Transactions on Signal Processing*, vol. 68, pp. 4050–4065, Jul. 2020.

[16] A. R. Flouro and S. P. Chadwick, "Crt-based deterministic reconstruction for sparse fft: First fully deterministic $o(k \log k)$ algorithm via gated chinese remainder theorem," *IEEE Transactions on Signal Processing*, 2025.

[17] A. R. Flouro and S. P. Chadwick, "Deterministic sparse fft via coprime decimation and iterative peeling: An $o(k \log n)$ algorithm with exact recovery guarantees," *IEEE Transactions on Signal Processing*, 2026.

[18] A. R. Flouro and S. P. Chadwick, "Keyed-gating sparse fft via structured residue views: Deterministic $o(\sqrt{N} \log k)$ frequency discovery," *IEEE Transactions on Signal Processing*, 2026.

[19] A. R. Flouro and S. P. Chadwick, "Deterministic regime selection for sparse spectrum sensing: Three-engine crossover analysis and production gate validation," 2025, arXiv preprint.

[20] RustFFT Contributors, "Rustfft: High-performance fft library written in pure rust," 2024, version 6.4.1. [Online]. Available: https://github.com/ejmahler/RustFFT

[21] R. J. Doviak and D. S. Zrnić, *Doppler Radar and Weather Observations*, 2nd ed. Mineola, NY, USA: Dover Publications, 2006.

[22] W. L. Melvin and J. A. Scheer, *Principles of Modern Radar: Radar Applications*. Raleigh, NC, USA: SciTech Publishing, 2014, vol. 3.

[23] H. D. Griffiths and C. J. Baker, *An Introduction to Passive Radar*. Norwood, MA, USA: Artech House, 2017.

[24] V. C. Chen and M. Martorella, *Inverse Synthetic Aperture Radar Imaging: Principles, Algorithms, and Applications*. Raleigh, NC, USA: SciTech Publishing, 2014.

[25] M. P. Hayes and P. T. Gough, "Synthetic aperture sonar: A review of current status," *IEEE Journal of Oceanic Engineering*, vol. 34, no. 3, pp. 207–224, Jul. 2009.

[26] D. L. Adamy, *EW 104: EW Against a New Generation of Threats*. Norwood, MA, USA: Artech House, 2015.

[27] J. Li and P. Stoica, *MIMO Radar Signal Processing*. Hoboken, NJ, USA: John Wiley & Sons, 2008.

[28] M. A. Richards, J. A. Scheer, and W. A. Holm, *Principles of Modern Radar: Basic Principles*. Raleigh, NC, USA: SciTech Publishing, 2010, vol. 1.

SparseTech Technical Papers