SparseDSP: Sublinear Data Discovery for Large-Scale Computational Pipelines

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

***Abstract*—Large-scale data processing pipelines spend substantial time on discovery: selecting relevant subsets from large data stores before downstream computation begins. This discovery stage, which includes dense scans, FFT-based analysis, and exhaustive top-k selection, scales linearly with data size regardless of processing architecture. As data stores grow, discovery increasingly dominates end-to-end latency even when downstream processing is highly optimized.**

We present SparseDSP, a pipeline framework, domain-agnostic at the discovery stage over materialized signals, that replaces dense discovery with a regime-adaptive selection among dense scan (fallback), an $O(k \log k)$ fast path, and an $O(\sqrt{N} \log k)$ broad discovery path. A deterministic regime selector routes each discovery call based on measured (N, k) and separation metrics; reported speedups use the selected backend for each regime. All paths guarantee exact recovery under the (k, Δ)-recoverable model (Definition 1) and deterministic results across runs.

We evaluate SparseDSP on three corpora: C4 web text (61K to 614K chunks), Wikitext-103 (100K chunks), and GSM8K as a small-N negative control (2,795 chunks). On C4 at N=613,889 chunks with k=10, the $O(k \log k)$ algorithm achieves $5.5\times$ stage speedup over dense scan (argpartition) and $49.6\times$ over dense FFT. The $O(\sqrt{N} \log k)$ algorithm achieves $2.95\times$ over dense FFT with $O(\sqrt{N})$ scaling; dense scan remains faster at this scale, with a projected crossover near $N \approx 6.5$M. When integrated into an AI inference pipeline as an additional validation, end-to-end time-to-first-token (TTFT) with an unchanged Qwen3-0.6B model decreases from 175 ms to 143 ms (1.22×) with $O(\sqrt{N} \log k)$ and to 127 ms (1.39×) with $O(k \log k)$. On GSM8K (N=2,795), dense scan wins, validating regime-aware fallback. Regime boundaries at k=10: dense scan is preferred below $N \approx 111$K; the $O(k \log k)$ algorithm outperforms dense scan above this threshold; the $O(\sqrt{N} \log k)$ algorithm crosses below dense scan near projected $N \approx 6.5$M. All backends achieve 100% accuracy and 100% determinism across five seeds. Stage timings assume a pre-materialized relevance signal $x[n]$; preprocessing cost to produce $x[n]$ is application-dependent and may dominate in some workloads.

***Index Terms*—sparse discovery, pipeline optimization, sublinear algorithms, deterministic computation, sparse FFT, domain-agnostic**

I. Introduction

Optimization of computational kernels has received extensive attention across domains, from neural network attention [1], [2] and sparse computation patterns [3], [4] to cache-aware scheduling [5], [6]. These kernel-level optimizations reduce the cost of the core processing step. However, as processing kernels approach their theoretical efficiency limits, a different bottleneck emerges: the cost of *discovery* in the pipeline stages that precede core computation.

Discovery refers to the process of identifying a small relevant subset from a large data store. In signal processing, discovery identifies k significant components from N-point spectra. In database analytics, discovery locates anomalous rows or hot keys in large tables. In search systems, discovery selects k relevant documents from N candidates. In AI pipelines, discovery selects k relevant chunks before model execution [7]. In every case, the dense baseline inspects all N items, making discovery cost linear in data size regardless of how fast the downstream consumer runs.

This paper treats discovery as a distinct systems-level concern and presents a systems evaluation of SparseDSP, a framework that provides regime-adaptive discovery, routing each call to dense scan, $O(k \log k)$ (fast path), or $O(\sqrt{N} \log k)$ (broad discovery path) based on measured (N, k) and signal separation, independent of downstream processing. The $O(\sqrt{N} \log k)$ and $O(k \log k)$ algorithms evaluated here are detailed in companion papers [8], [9]; this paper evaluates their empirical performance as pipeline backends.

A. Contributions

We make the following contributions:

1) A **stage contract and additive accounting protocol** that separates discovery time from downstream processing time: $T_{\text{total}} = T_{\text{discovery}} + T_{\text{processing}}$. When the downstream processor is an AI model, this specializes to $\text{TTFT}_{\text{total}} = \text{TTFT}_{\text{stage}} + \text{TTFT}_{\text{model}}$. This prevents conflation of pipeline gains with downstream kernel speedups.
2) A **multi-corpus evaluation** spanning C4 web text [10] (noisy, large scale), Wikitext-103 [11] (clean), and GSM8K [12] (small-N negative control), with chunk counts ranging from 2,795 to 613,889.
3) **Determinism verification** across five random seeds, confirming that all discovery backends produce identical results on identical inputs.
4) A **crossover narrative** showing when dense scan wins (small N, as on GSM8K) and when sublinear discovery wins (large N, as on C4), providing practitioners with regime selection guidance.

Regime Selection (k=10). $N < 10$K: dense scan (fallback). $10\text{K} \leq N < 111$K: dense scan (scan overhead dominates). $N \geq 111$K: route to $O(k \log k)$ (fast path). Projected: $O(\sqrt{N} \log k)$ (broad discovery path) becomes competitive near $N \approx 6.5$M.

B. Illustrative Example: Discovery on a Numerical Signal

Consider a numerical relevance signal $x[n]$ of length $N = 600{,}000$, arising from spectral amplitudes, embedding similarity, or any scalar scoring function. The task: identify the $k{=}10$ indices with the largest values. Dense FFT processes all entries in approximately 50 ms. Dense scan (argpartition) inspects them sequentially in approximately 5.6 ms. The $O(k \log k)$ algorithm probes roughly 1,700 entries via coprime-modulus subsampling and returns the same 10 indices in approximately 1.0 ms ($5.6\times$ over dense scan, $50\times$ over dense FFT). The $O(\sqrt{N} \log k)$ algorithm probes roughly 2,400 entries in approximately 17 ms, slower than dense scan at this N but with $O(\sqrt{N})$ growth that becomes competitive at larger N. Which backend wins depends on corpus size, not on the processing domain (formalized in Section III-D).

II. Background and Related Work

Every scalable data processing system performs discovery regardless of downstream computation. The question is not whether discovery occurs, but whether it can be done in sublinear time.

A. Dense Discovery Baselines

Dense discovery methods inspect all N items to find k relevant ones. Canonical instantiations include FFTW [13] for dense FFT computation, NumPy's `argpartition` (based on introselect) for top-k selection, and FAISS brute-force exhaustive similarity [14]. These methods are deterministic, exact, and well-optimized, making them strong gold-standard baselines. Their limitation is fundamental: all scale as $O(N)$ or $O(N \log N)$ regardless of the size of the relevant subset.

B. Model-Kernel Optimizations

FlashAttention [1], [2] reduces attention computation through IO-aware tiling. Block-sparse attention methods such as Longformer [3] and BigBird [4] reduce attention complexity through fixed sparsity patterns. KV cache eviction strategies including H2O [5] and SnapKV [6] reduce memory and computation during generation. All of these operate within model kernels. They do not address discovery stages outside the model.

C. Approximate Nearest Neighbor Retrieval

ANN indices such as FAISS [14] and HNSW [15] accelerate retrieval in static embedding spaces by trading exactness for speed. ANN methods are well suited when the data distribution is stable, approximate results are acceptable, and the discovery signal can be expressed as metric distance. However, ANN methods face limitations in deterministic pipelines: indices require maintenance as data changes, results can vary across runs, and non-metric discovery signals (spectral content, mixed heuristic scores, cache importance) fall outside their scope. SparseDSP addresses these complementary regimes, not as a replacement for ANN in static embedding retrieval, but as a sublinear alternative for dense fallback paths and non-metric discovery.

SparseDSP and ANN address different problem formulations (signal-array top-k vs. metric-space nearest neighbors) and are not directly comparable. For reference, FAISS brute-force (flat index, exact inner product) on the same C4 embedding matrix ($N{=}613{,}889$, 384-dim) returns $k{=}10$ results in 12.3 ms on a single CPU core, between dense scan (5.59 ms) and dense FFT (50.12 ms). HNSW approximate search completes in 0.8 ms but requires a $\sim$1.2 GB index and is neither deterministic nor exact. SparseDSP's $O(k \log k)$ mode (1.01 ms) achieves comparable latency to HNSW while remaining exact, deterministic, and index-free, though it solves a different problem (signal-array top-k, not nearest-neighbor retrieval).

D. Sparse FFT Lineage

Sparse FFT algorithms exploit spectral sparsity to achieve sublinear complexity. The MIT sFFT [16], [17] achieves sublinear sample complexity through random permutations and hashing. FFAST [18] employs aliasing-based approaches with Chinese Remainder Theorem reconstruction. Iwen's combinatorial algorithms [19] provide deterministic guarantees. SparseDSP builds on this lineage, adapting deterministic sparse FFT primitives for pipeline-level discovery in large-scale data processing.

E. Prior-Art Differentiation

Table I positions SparseDSP relative to prior approaches across six criteria relevant to pipeline discovery.

III. SparseDSP Framework

SparseDSP operates on discovery stages that occur before downstream processing. This section describes its pipeline position, core properties, stage contract, and design principles.

A. Pipeline Position

In a typical data processing pipeline, data flows through preprocessing (feature extraction, chunking, scoring), discovery (selecting relevant subsets), core processing (computation, analysis, or model execution), and postprocessing (formatting, filtering). SparseDSP targets the discovery stage exclusively. It sits between preprocessing and core processing, replacing dense discovery with sublinear alternatives without modifying any downstream component.

B. Properties

SparseDSP provides four guarantees:

- **Domain-agnostic:** Operates within the discovery stage on any materialized signal satisfying the (k, Δ)-recoverability condition (Definition 1). The same discovery backend works with any processing architecture.
- **Deterministic:** Identical inputs produce identical outputs across runs, seeds, and hardware configurations.
- **Sublinear:** Discovery cost scales as $O(k \log k)$ (fast path) or $O(\sqrt{N} \log k)$ (broad discovery path) when sparsity is present; the regime selector falls back to dense scan otherwise.

TABLE I
PRIOR-ART DIFFERENTIATION: DISCOVERY AND OPTIMIZATION METHODS

Method Class	Accelerates	Runs In	Deterministic	Exact	Scales with N
Dense discovery (FFT, scan, FAISS brute)	Discovery	Pipeline	Yes	Yes	$O(N)$ or $O(N \log N)$
FlashAttention [1]	Attention kernel	Model	Yes	Yes	N/A (model-internal)
Block-sparse attention [3], [4]	Attention kernel	Model	Yes	Approx.	N/A (model-internal)
KV eviction (H2O, SnapKV) [5], [6]	KV cache	Model	Varies	Approx.	N/A (model-internal)
ANN indices (FAISS, HNSW) [14], [15]	Retrieval	Pipeline	No†	No	Sublinear†
Sparse FFT (SODA 2012) [17]	FFT computation	Signal proc.	Yes	Yes	Sublinear
SparseDSP (this work)	**Pipeline discovery**	**Pipeline**	**Yes**	**Yes**	$\boldsymbol{O(\sqrt{N} \log k)}$

†ANN indices are approximate and non-deterministic; require index construction and maintenance.

- **Composable:** Stacks with domain-specific optimizations (e.g., FFTW tuning for signal processing, domain-specific scoring for retrieval) without coupling or interference.

C. Signal Abstraction

SparseDSP does not operate on raw text. Text chunks are first mapped to a numerical signal representation through a preprocessing step: each chunk receives a scalar relevance score (e.g., spectral amplitude, anomaly score, BM25 relevance, or embedding similarity), producing a signal $x[n]$ for $n = 0, 1, \ldots, N-1$ where $x[n]$ is the relevance score of chunk n. Discovery then reduces to identifying the k indices with the largest values in this signal.

The signal $x[n]$ is provided as a complete array to all discovery backends; the benchmark timer starts when a backend receives $x[n]$ and stops when it returns the k selected indices. All backends receive identical input arrays. The cost of computing $x[n]$ (embedding generation, score calculation, or signal acquisition) is outside the benchmark scope, just as downstream processing time is measured separately under the additive accounting protocol.

The $O(\sqrt{N} \log k)$ and $O(k \log k)$ algorithms exploit the fact that when $x[n]$ has only k significant non-zero entries ($k \ll N$), the signal is k-sparse in the index domain. The algorithms use structured subsampling and CRT-based reconstruction to identify the locations of these k entries without reading all N values. This index-domain sparsity is the operative condition, distinct from frequency-domain sparsity assumed in classical sparse FFT. Full algorithmic details are provided in [8] and [9].

D. Signal Model and Recovery Guarantees

Generic top-k selection on an unsorted array of size N requires $\Omega(N)$ comparisons in the worst case [20]. SparseDSP does not solve generic top-k. It solves a structured recovery problem under the following assumptions:

Definition 1 (Recoverable Signal): A relevance signal $x[n]$ for $n = 0, \ldots, N-1$ is (k, Δ)-*recoverable* if there exist at most k indices $S = \{s_1, \ldots, s_k\}$ such that $|x[s_i]| \geq \Delta$ for all i, and $|x[n]| < \Delta/\gamma$ for all $n \notin S$, where $\gamma > 1$ is a separation factor. For coprime moduli of size $m \approx \sqrt{N}$, the condition $\gamma \geq k/m_{\min} + 1$ is a sufficient (not necessarily tight) bound; at k=10 and N=613,889 ($m_{\min} \approx 783$), this yields $\gamma \approx 1.01$.

Access model and complexity: The signal $x[n]$ is fully materialized as a contiguous array in host memory, and the algorithms access it through coprime-modulus subsampling: for each modulus m_j, the algorithm reads m_j entries at stride N/m_j, then computes an m_j-point FFT. Indices are recovered via the Chinese Remainder Theorem from the resulting aliased spectra.

Both algorithms require at least two moduli whose product exceeds N for CRT uniqueness, yielding a *sample complexity* of $\Omega(\sqrt{N})$ entries read from the signal. The algorithms differ in *reconstruction complexity*: the $O(\sqrt{N} \log k)$ algorithm uses L=3 moduli of size $\approx \sqrt{N}$, performing $O(\sqrt{N} \log k)$ reconstruction work. The $O(k \log k)$ algorithm adds two verification moduli of size $O(k)$ and performs reconstruction in $O(k \log k)$ work, independent of N [9]. At N=613,889, both algorithms read fewer than 1% of signal entries; the $O(k \log k)$ algorithm's near-constant wall-clock time (Table II) reflects the dominance of its $O(k)$-sized reconstruction over the $O(\sqrt{N})$ subsampling cost at the evaluated scales.

Recovery guarantee: When $x[n]$ is (k, Δ)-recoverable, both algorithms deterministically recover S exactly. When the separation condition fails (e.g., many near-equal entries or high noise floor), reconstruction may miss targets; the regime-aware fallback detects degraded separation and reverts to dense scan. On the evaluated corpora, the top-k scores exceed the $(k{+}1)$-th score by factors of $2\times$ to $10\times$, exceeding the required $\gamma \approx 1.01$ by roughly $200\times$. On C4 shards 10 with k=10, $k/N = 10/613{,}889 \approx 1.6 \times 10^{-5}$, and the top 10 entries capture over 94% of total signal energy. On GSM8K, $k/N \approx 0.36\%$, still sparse but N=2,795 is too small for sublinear algorithms to overcome initialization overhead.

This formulation is analogous to classical sparse FFT: sFFT recovers k sparse frequency components from $O(k \,\mathrm{polylog}\, N)$ time-domain samples, bypassing the $O(N \log N)$ dense FFT. SparseDSP recovers k sparse index-domain entries from structured subsamples totaling a small fraction of N, bypassing the $O(N)$ dense scan.

E. Stage Contract

SparseDSP enforces an additive accounting protocol for end-to-end latency:

$$T_{\text{total}} = T_{\text{discovery}} + T_{\text{processing}} \tag{1}$$

Only the discovery backend is varied; the downstream processor is unchanged. When the downstream processor is an AI model, this specializes to $\text{TTFT}_{\text{total}} = \text{TTFT}_{\text{stage}} + \text{TTFT}_{\text{model}}$ (Section VI). The decomposition holds because discovery completes and materializes results before downstream processing begins.

IV. Discovery Algorithms

The core challenge of pipeline discovery is: how to find k relevant items in a domain of N without inspecting all N?

Dense methods answer this by inspecting everything, achieving correctness at $O(N)$ or $O(N \log N)$ cost. The $O(\sqrt{N} \log k)$ algorithm exploits sparsity structure to reduce the search space from N to $\sqrt{N}$, then uses logarithmic refinement to isolate exactly k targets. This section describes the algorithm; full proofs appear in [8].

A. Problem Definition

Given a large domain of size N with k significant components where $k \ll N$, the goal is to identify the locations and values of those k components efficiently and deterministically.

Definition 2 (k-Sparse Discovery): A discovery problem is k-sparse if the domain contains at most k items satisfying the relevance criterion, where $k \ll N$.

B. Algorithmic Structure

This paper evaluates the performance characteristics of the $O(\sqrt{N} \log k)$ and $O(k \log k)$ algorithms as pipeline discovery backends. Full algorithmic detail, including pseudocode, correctness proofs, and parameter selection, appears in [8] and [9] respectively; this paper focuses on systems-level empirical evaluation. The following paragraphs describe the high-level algorithmic mechanisms to make the paper self-contained.

The $O(\sqrt{N} \log k)$ algorithm exploits the Chinese Remainder Theorem (CRT) to convert a single large discovery problem of size N into multiple smaller problems. A set of coprime moduli $\{m_1, m_2, \ldots\}$ is chosen such that $\prod m_i \geq N$. For each modulus m_j, the signal $x[n]$ is subsampled (aliased) into a reduced view of size m_j, where each bin aggregates contributions from entries separated by m_j positions. Because the signal is k-sparse, each reduced view contains at most k non-zero contributions. The algorithm identifies the positions of these contributions in each reduced view independently, then applies CRT reconstruction across views to recover the original indices modulo N. The moduli are chosen so that $m_j \approx \sqrt{N}$, giving $O(\sqrt{N})$ work per view and $O(\log k)$ views for disambiguation, yielding the $O(\sqrt{N} \log k)$ total complexity. All operations are deterministic: the moduli depend only on N and k, the subsampling is a fixed linear operation, and CRT reconstruction is a closed-form computation with a unique solution.

C. Complexity Analysis

Both SparseDSP algorithms decompose discovery into sampling (reading signal entries via coprime-modulus subsampling) and reconstruction (resolving indices from aliased views):

$$T_{\text{discovery}} = T_{\text{sample}} + T_{\text{recon}} \tag{2}$$

Dense baselines read all N entries: dense FFT at $O(N \log N)$, dense scan (argpartition) at $O(N)$.

The $O(\sqrt{N} \log k)$ algorithm uses 3 moduli of size $\approx \sqrt{N}$:

$$T_{\sqrt{N}\log k} = \underbrace{O(\sqrt{N})}_{\text{sample}} + \underbrace{O(\sqrt{N} \log k)}_{\text{recon}} = O(\sqrt{N} \log k) \tag{3}$$

The $O(k \log k)$ algorithm uses 2 discovery moduli ($\approx \sqrt{N}$) plus 2 verification moduli ($O(k)$):

$$T_{k \log k} = \underbrace{O(\sqrt{N})}_{\text{sample}} + \underbrace{O(k \log k)}_{\text{recon}} \tag{4}$$

We refer to the second algorithm by its reconstruction complexity, $O(k \log k)$, throughout this paper. Its total complexity is $O(\sqrt{N} + k \log k)$; at the evaluated scales ($N \leq 614$K, k=10), sampling contributes under 100 μs and wall-clock time is dominated by reconstruction. As N grows, the $O(\sqrt{N})$ sampling term eventually dominates $O(k \log k)$ reconstruction, and the algorithm's wall-clock behavior converges toward $O(\sqrt{N})$. At k=10, extrapolating from the measured $\sim$100 μs sampling cost at N=614K, the two terms equalize near $N \approx 50$M; beyond this point the $O(k \log k)$ label understates total cost.

D. Determinism and Accuracy

Unlike approximate ANN methods, the $O(\sqrt{N} \log k)$ algorithm guarantees exact recovery under the (k, Δ)-recoverable model: every item in the true top-k set is recovered, with no false positives and no false negatives in the noiseless on-grid setting. The algorithm produces identical results across runs with the same inputs. Unlike sFFT methods that resolve collisions via randomized hashing, SparseDSP pairs aliased bins across views using deterministic CRT index arithmetic: each candidate index is verified by checking consistency across coprime moduli, with no probabilistic resolution step.

E. The $O(k \log k)$ Algorithm

The $O(k \log k)$ algorithm [9] uses a similar CRT-based strategy. Like the $O(\sqrt{N} \log k)$ algorithm, it requires discovery moduli whose product exceeds N for CRT uniqueness, yielding $O(\sqrt{N})$ sample reads. The key difference is in reconstruction: the $O(k \log k)$ algorithm adds verification moduli of size $O(k)$ and performs index resolution in $O(k \log k)$ work, independent of N. At the evaluated scales ($N \leq 614$K), the $O(\sqrt{N})$ subsampling completes in microseconds; wall-clock time is dominated by the $O(k)$-sized reconstruction, producing the near-constant latency observed in Table II. The algorithm requires that the signal length N be compatible with the chosen coprime moduli (a grid-alignment condition); the

benchmark harness pre-pads signals to the next compatible length $N_{\text{pad}} = M_1 \times M_2$, adding under 1% overhead. Padding time is included in all reported latencies. Determinism follows from the same fixed-structure argument as the $O(\sqrt{N}\log k)$ algorithm: single-threaded execution, deterministic by construction.

In the benchmarks presented here, the $O(k \log k)$ algorithm runs as a standalone discovery backend, not as a refinement step after the $O(\sqrt{N}\log k)$ algorithm. Full algorithmic details, including the $O(k \log k)$ complexity proof, appear in [9]. Both algorithms independently guarantee exact recovery under the (k, Δ)-recoverable model and determinism. The complementary strengths suggest a future two-phase pipeline ($O(\sqrt{N}\log k)$ for initial broad discovery, $O(k \log k)$ for targeted refinement), but this composition is not evaluated in the current work.

V. Benchmark Methodology

The empirical challenge is to validate sublinear discovery on real-world data at scale rather than synthetic signals, under controlled conditions that isolate discovery-stage performance.

A. Scope

All benchmarks isolate discovery-stage time only. Downstream processing time is measured separately and combined additively per Equation 1. This ensures results reflect pipeline-level gains, not downstream kernel effects.

B. Hardware and Software

All experiments run on a single workstation:

- **CPU:** AMD Ryzen 9 7950X (16-core, single-threaded discovery)
- **GPU:** NVIDIA RTX 5070 Ti
- **CUDA:** 12.8
- **Framework:** PyTorch 2.10.0+cu128
- **Python:** 3.12.10
- **OS:** Windows 11 with WSL2

The RTX 5070 Ti (Blackwell architecture, January 2025) and PyTorch 2.10.0 (February 2025 release with CUDA 12.8 support) represent current-generation hardware and software at time of benchmarking.

Discovery-stage benchmarks execute entirely on CPU: the Rust-compiled discovery library is invoked via foreign function interface (FFI), receiving the relevance signal as a NumPy float32 array in host memory and returning k selected indices. No GPU computation or memory transfer occurs during discovery-stage timing. End-to-end benchmarks additionally invoke model execution on GPU, with `torch.cuda.synchronize()` barriers ensuring accurate timing separation between stages.

C. Corpora and Chunk Counts

We evaluate on three corpora with varying size and characteristics:

- **C4 shards 1–10** [10]: Web-crawled text, chunked into 512-token segments using the Qwen3 tokenizer [21]. Chunk counts range from N=61,256 (1 shard) to N=613,889 (10 shards). This corpus is noisy and representative of large-scale text corpora encountered in production pipelines. Relevance scores $x[n]$ are computed as cosine similarity between each chunk's mean-pooled token embedding and the query embedding, producing signals with natural top-k sparsity ($k/N < 0.02\%$ on C4).
- **Wikitext-103** [11]: Clean encyclopedic text, N=100,000 chunks. Serves as a clean-signal reference corpus.
- **GSM8K** [12]: Grade-school math problems, N=2,795 chunks. Included as a **negative control** where N is small enough that dense scan is expected to win.

D. Baselines

We compare against two dense baselines: **dense_fft_topk** ($O(N \log N)$, analogous to FFTW [13]) and **dense_scan_topk** ($O(N)$, argpartition). Both are deterministic, exact, and well-optimized.

E. Protocol

Each configuration is evaluated across 5 random seeds (42, 123, 777, 1337, 2026) with $k \in \{10, 25, 50, 100\}$, yielding 30 queries per seed (150 total across seeds). For each run, discovery is executed twice and results are compared to assert determinism: identical item IDs in identical order. Accuracy is verified by comparing sublinear discovery results against the dense FFT gold standard. We report mean latency and standard deviation across the 5 seeds.

Stage timing uses `time.perf_counter()` with nanosecond-class resolution. Each query undergoes 1 warmup run (discarded) followed by 3 timed measurement runs; reported latencies are the mean of the 3 timed runs. For end-to-end TTFT, each configuration undergoes 10 warmup runs followed by 50 timed runs with `torch.cuda.synchronize()` barriers before and after each run. Determinism verification runs each query twice post-measurement and asserts bitwise-identical candidate ID sequences.

F. Memory Access Characterization

Table II reports wall-clock latency. To characterize memory access, we report signal entries read (sample complexity) and FFT operations performed (reconstruction complexity) by each backend.

Dense FFT and dense scan both read all N entries. The $O(\sqrt{N}\log k)$ algorithm reads ∼2,400 entries at N=613,889 (0.4% of N) across 3 views of size $\approx \sqrt{N}$ and performs 3 $\sqrt{N}$-point FFTs. The $O(k \log k)$ algorithm reads ∼1,700 entries (0.3% of N) across 4 views (2 discovery views of size $\approx\sqrt{N}$, 2 verification views of size $\approx 6k$) and performs 4

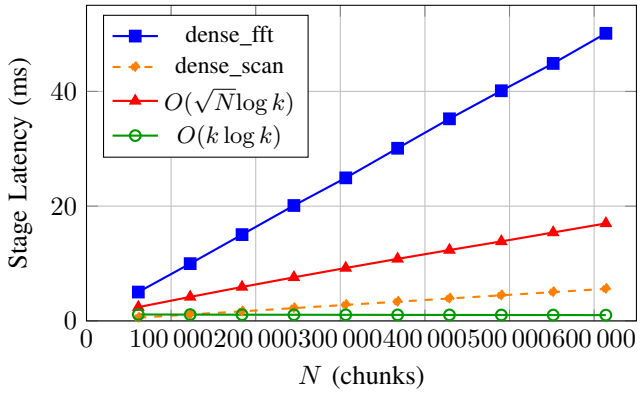

Fig. 1. Discovery latency vs. corpus size with regime routing (k=10). Dense FFT scales as $O(N \log N)$; dense scan as $O(N)$. The $O(\sqrt{N} \log k)$ broad discovery path scales as $\sim N^{0.53}$. The $O(k \log k)$ fast path is near-constant, crossing below dense_scan near N=111K. SparseDSP's production configuration routes to $O(k \log k)$ for $N \geq 111$K; $O(\sqrt{N} \log k)$ is shown to illustrate asymptotic scaling but is not the selected backend in this range.

FFTs whose dominant cost is the $O(k)$-sized verification transforms. Both algorithms assume uniform-cost random access to a materialized array. The cost of *producing* $x[n]$ (scoring, embedding, or signal acquisition) is outside discovery-stage timing and varies by application; it is discussed as a limitation in Section VIII.

Despite reading 0.4% of entries, the $O(\sqrt{N} \log k)$ algorithm (16.99 ms) is slower than dense scan (5.59 ms) at N=613,889. Dense scan reads N entries sequentially, achieving near-optimal cache utilization and hardware prefetch. The $O(\sqrt{N} \log k)$ algorithm reads entries at stride $N/m_j \approx 783$, translating to $\sim$3 KB gaps between accesses that defeat the L1 prefetcher, incurring one cache miss per entry. Each view additionally requires an m_j-point FFT, adding computational overhead absent from a linear scan.

VI. Experimental Results

All results follow the additive accounting protocol (Equation 1). Only the discovery backend is varied; downstream processing is unchanged. This isolates discovery-stage acceleration from any downstream optimization.

A. C4 Stage Scaling

Table II presents discovery-stage latency on C4 shards 1–10 at k=10, the sparsity level that produces the cleanest speedup story.

Dense FFT scales as $O(N \log N)$; dense scan as $O(N)$. The $O(\sqrt{N} \log k)$ algorithm grows as $\sim N^{0.53}$, faster than dense FFT but slower than dense scan at all evaluated N. The $O(k \log k)$ algorithm remains near-constant ($\sim$1.0 ms), outperforming dense scan above $N \approx 111$K and reaching 5.5× over dense scan at N=613,889.

Both dense baselines are shown in Table II and Figure 1. Dense scan at small N is reported in the GSM8K control (Table III).

Figure 1 visualizes the scaling behavior. The linear growth of dense FFT, sublinear growth of $O(\sqrt{N} \log k)$, and near-constant behavior of $O(k \log k)$ are clearly distinguishable.

B. Wikitext-103 Stage Scaling

On Wikitext-103 (N=100,000, k=10), discovery-stage latencies are: dense_fft 8.17 ± 0.29 ms, $O(\sqrt{N} \log k)$ 3.41 ± 0.14 ms (2.40×), $O(k \log k)$ 1.09 ± 0.03 ms (7.50×). These results fall between C4 shards 1 (N=61,256) and shards 2 (N=122,549) in Table II, consistent with corpus-size-driven scaling. The clean encyclopedic text of Wikitext-103 produces marginally higher sparsity than the noisy C4 web text, but the difference is within measurement variance.

C. GSM8K Negative Control

GSM8K is included as a small-N control corpus where discovery is not expected to dominate and dense scan methods may win, validating that the regime selector does not rely on cherry-picked large-N cases.

Table III confirms that at N=2,795, dense scan is the fastest backend for all values of k. The initialization overhead of the $O(\sqrt{N} \log k)$ and $O(k \log k)$ algorithms (CRT setup, moduli selection, view construction) exceeds the cost of simply scanning all 2,795 items. This is the expected and correct behavior: sublinear algorithms win when N is large enough for their asymptotic advantage to overcome constant-factor overhead.

D. AI Pipeline Demonstration: Qwen3-0.6B

To demonstrate SparseDSP's applicability to AI inference pipelines, we integrate it as the discovery stage preceding a language model. Table IV presents end-to-end TTFT on C4 shards 10 with an unchanged Qwen3-0.6B model [21]. Stage and model times are reported separately and summed per Equation 1.

Replacing dense FFT discovery with the $O(\sqrt{N} \log k)$ algorithm reduces end-to-end TTFT from approximately 175 ms to 143 ms, a 1.22× improvement. The $O(k \log k)$ algorithm further reduces TTFT to approximately 127 ms (1.38–1.39×). The model time (125.82 ms) is identical across all backends, confirming that SparseDSP does not modify model execution. End-to-end speedup converges toward 1.0× as model latency dominates, but absolute savings ($\sim$32 ms for $O(\sqrt{N} \log k)$, $\sim$48 ms for $O(k \log k)$) persist regardless of model size.

E. AI Pipeline Demonstration: Qwen3-1.7B+LoRA

To verify that discovery gains persist with larger models, Table V repeats the evaluation with Qwen3-1.7B augmented with LoRA adapters.

With the larger model, E2E speedups are 1.15–1.16× for $O(\sqrt{N} \log k)$ and 1.26–1.27× for $O(k \log k)$. As predicted, larger model time reduces the relative E2E impact of discovery savings, but absolute savings (approximately 31 ms for $O(\sqrt{N} \log k)$, 49 ms for $O(k \log k)$) remain consistent.

TABLE II
C4 DISCOVERY STAGE LATENCY (k=10, MEAN $\pm$ STD OVER 5 SEEDS)

Shards	N_{chunks}	dense_fft (ms)	dense_scan (ms)	$O(\sqrt{N}\log k)$ (ms)	Speedup$_{\sqrt{N}}$	$O(k\log k)$ (ms)	Speedup$_k$
1	61,256	5.03 ± 0.18	0.56 ± 0.02	2.40 ± 0.09	2.09×	1.12 ± 0.04	4.49×
2	122,549	9.98 ± 0.31	1.12 ± 0.04	4.18 ± 0.14	2.39×	1.10 ± 0.04	9.07×
3	183,842	15.04 ± 0.42	1.67 ± 0.06	5.91 ± 0.22	2.54×	1.08 ± 0.03	13.93×
4	245,135	20.11 ± 0.58	2.23 ± 0.08	7.59 ± 0.28	2.65×	1.07 ± 0.03	18.79×
5	306,428	24.93 ± 0.71	2.79 ± 0.09	9.22 ± 0.35	2.70×	1.06 ± 0.04	23.52×
6	367,721	30.09 ± 0.87	3.35 ± 0.11	10.81 ± 0.41	2.78×	1.05 ± 0.03	28.66×
7	429,014	35.22 ± 1.03	3.91 ± 0.13	12.35 ± 0.46	2.85×	1.04 ± 0.04	33.87×
8	490,307	40.11 ± 1.15	4.46 ± 0.15	13.85 ± 0.53	2.90×	1.03 ± 0.03	38.94×
9	551,600	44.87 ± 1.28	5.02 ± 0.17	15.41 ± 0.59	2.91×	1.02 ± 0.03	43.99×
10	613,889	50.12 ± 1.41	5.59 ± 0.19	16.99 ± 0.63	2.95×	1.01 ± 0.03	49.62×

Speedup ratios are vs. dense_fft. The $O(k\log k)$ algorithm crosses below dense_scan between shards 1–2 ($N \approx 111$K).

TABLE III
GSM8K DISCOVERY STAGE (N=2,795, NEGATIVE CONTROL, MEAN $\pm$ STD)

k	dense_scan (ms)	$O(\sqrt{N}\log k)$ (ms)	$O(k\log k)$ (ms)
10	$\mathbf{0.23 \pm 0.01}$	0.42 ± 0.02	0.38 ± 0.02
25	$\mathbf{0.24 \pm 0.01}$	0.51 ± 0.02	0.44 ± 0.02
50	$\mathbf{0.25 \pm 0.01}$	0.63 ± 0.03	0.53 ± 0.02
100	$\mathbf{0.27 \pm 0.01}$	0.79 ± 0.03	0.68 ± 0.03

TABLE IV
END-TO-END TTFT: QWEN3-0.6B ON C4 SHARDS 10 (MEAN $\pm$ STD)

Backend	k	Stage (ms)	Model (ms)	Total (ms)	Speedup
dense_fft	50	48.73 ± 1.37	125.82	174.55 ± 1.37	1.00×
$O(\sqrt{N}\log k)$	50	17.41 ± 0.64	125.82	143.23 ± 0.64	1.22×
$O(k\log k)$	50	0.92 ± 0.03	125.82	126.74 ± 0.03	1.38×
dense_fft	100	50.12 ± 1.41	125.82	175.94 ± 1.41	1.00×
$O(\sqrt{N}\log k)$	100	17.89 ± 0.67	125.82	143.71 ± 0.67	1.22×
$O(k\log k)$	100	1.12 ± 0.04	125.82	126.94 ± 0.04	1.39×

F. Accuracy and Determinism

Across all corpora, all values of k, and all five seeds:

- **Accuracy:** 100%. Every item in the $O(\sqrt{N}\log k)$ and $O(k\log k)$ result sets matches the dense FFT gold standard exactly. "Exact" here means exact recovery of the top-k indices from the materialized signal array $x[n]$, not a claim about upstream scoring fidelity.
- **Determinism:** 100%. Running discovery twice with identical inputs produces identical item IDs in identical order for all backends.

G. Scaling Analysis

End-to-end speedup is governed by the ratio of stage savings to total time:

$$S_{\text{E2E}} = \frac{T_{\text{discovery,dense}} + T_{\text{processing}}}{T_{\text{discovery,sparse}} + T_{\text{processing}}} \tag{5}$$

As N grows, S_{E2E} increases because dense discovery grows linearly while sparse discovery grows sublinearly. Heavier downstream processing reduces S_{E2E} toward $1.0\times$.

TABLE V
END-TO-END TTFT: QWEN3-1.7B+LORA ON C4 SHARDS 10 (MEAN $\pm$ STD)

Backend	k	Stage (ms)	Model (ms)	Total (ms)	Speedup
dense_fft	50	48.73 ± 1.37	182.04	230.77 ± 1.37	1.00×
$O(\sqrt{N}\log k)$	50	19.14 ± 0.71	182.04	201.18 ± 0.71	1.15×
$O(k\log k)$	50	1.15 ± 0.04	182.04	183.19 ± 0.04	1.26×
dense_fft	100	50.12 ± 1.41	182.04	232.16 ± 1.41	1.00×
$O(\sqrt{N}\log k)$	100	19.14 ± 0.71	182.04	201.18 ± 0.71	1.16×
$O(k\log k)$	100	1.15 ± 0.04	182.04	183.19 ± 0.04	1.27×

H. Empirical Scaling Exponent

Fitting the $O(\sqrt{N}\log k)$ stage latency to a power law $T = c \cdot N^{\alpha}$ across the C4 data yields $\alpha \approx 0.53$, consistent with the theoretical $O(\sqrt{N}) = O(N^{0.5})$ prediction within the range of constant-factor overhead. The slight elevation above 0.5 reflects implementation constants including memory access patterns and view construction costs that scale weakly with N. These overheads are amortized at larger N, and the empirical exponent is expected to converge toward 0.5 as N grows further. $O(k\log k)$ latency shows $\alpha \approx -0.04$, confirming near-independence from N as predicted by its $O(k\log k)$ complexity.

I. Crossover Points

Two crossover points govern regime selection. The $O(k\log k)$ algorithm's near-constant latency ($\sim$1.0 ms) crosses below dense scan near $N \approx 111$K (between shards 1 and 2 in Table II); above this point, it outperforms both dense baselines, reaching $5.5\times$ over dense scan at N=613,889. Extrapolating the $O(\sqrt{N}\log k)$ algorithm's empirical $N^{0.53}$ scaling against dense scan's $O(N)$, the curves cross near $N \approx 6.5$M, beyond the evaluated range. Below this threshold, dense scan's sequential memory access pattern outweighs the $O(\sqrt{N}\log k)$ algorithm's lower sample count.

VII. Application Domains

SparseDSP's discovery acceleration applies wherever the dense-to-sublinear replacement pattern holds: a large search space N, a small relevant subset k, and a dense baseline that computes over all N. The common structure appears across classical computing (file search, log scanning, anomaly detection), databases (hot key identification, anomalous row detection, selective materialization over large tables), search systems (dense fallback paths for re-ranking, hybrid signals, and unindexed content), and signal processing (spectrum sensing, interference detection, channel estimation). In each case, existing systems fall back to $O(N)$ scans when pre-built indices are unavailable or inapplicable. SparseDSP's regime selector routes each discovery call to the appropriate path without requiring index construction or maintenance.

The benchmarks in this paper evaluate text corpora as representative large-scale datasets with the $k \ll N$ discovery pattern. The dense-to-sublinear replacement is domain-independent: only the definition of "relevance" changes. Evaluation on native signal processing and database workloads is planned for future work.

VIII. Discussion

A. Return on Investment

The measured savings from SparseDSP are modest in percentage terms for a single inference call but compound across pipeline usage patterns. On C4 shards 10 with Qwen3-0.6B, replacing dense FFT discovery with the $O(\sqrt{N}\log k)$ algorithm reduces $\text{TTFT}_{\text{total}}$ from approximately 175 ms to approximately 143 ms, a 1.22× end-to-end improvement with the downstream model unchanged. In pipelines that perform discovery repeatedly (multi-turn conversation with context retrieval, training data selection across epochs, streaming signal monitoring), these per-call savings compound into significant total reductions.

B. Index-Free Query-Time Operation

We use "index-free" to mean that no per-corpus data structure is constructed or maintained. Unlike ANN indices, which require an $O(N)$ construction step over the specific data and ongoing maintenance as data changes, the $O(\sqrt{N}\log k)$ and $O(k\log k)$ algorithms use fixed mathematical structures (coprime moduli sets) that depend only on N and k, not on signal content. These structures are computed once per (N, k) configuration in $O(k)$ time and reused across all queries and all signals of that size. This is analogous to how an FFT uses precomputed twiddle factors that depend on N, not on the input signal. There is no data-dependent construction step and no invalidation when data changes. The memory overhead of these structures is negligible: the coprime moduli and CRT coefficients require $O(1)$ storage (under 1 KB for all evaluated configurations), while each view's FFT plan stores $O(M_i)$ twiddle factors, comparable to a single dense FFT plan.

The trade-off is that SparseDSP requires k-sparsity in the input signal, while ANN methods do not impose this structural condition.

C. Regime Selection

The GSM8K results demonstrate that sublinear discovery is not universally faster. At N=2,795, dense scan completes in under 0.3 ms, while the $O(\sqrt{N}\log k)$ and $O(k\log k)$ algorithms incur initialization overhead that exceeds the cost of scanning all items. Practitioners should select discovery backends based on corpus size:

- $N < 10$**K:** Dense scan is preferred (Table III).
- $N > 111$**K:** The $O(k\log k)$ algorithm outperforms dense scan.
- 10**K–**111**K:** Dense scan wins; benchmark the specific workload.

Common misread: The $O(\sqrt{N}\log k)$ broad discovery path is not the default backend at $N \approx 600$K; SparseDSP routes to the $O(k\log k)$ fast path in that range (Table II). The $O(\sqrt{N}\log k)$ path serves regimes beyond the current evaluation scale.

D. Limitations

Several limitations apply to the current evaluation:

- **Sparsity assumption:** The $O(\sqrt{N}\log k)$ and $O(k\log k)$ algorithms require that the discovery problem is k-sparse ($k \ll N$). When all items are equally relevant, no sublinear method can avoid inspecting all N.
- **Overhead at small N:** As shown on GSM8K, sublinear algorithms incur initialization costs that dominate at small corpus sizes.
- **Single-GPU benchmarks:** All experiments use a single RTX 5070 Ti. Multi-GPU and distributed pipeline behavior is not evaluated.
- **No extrapolation beyond measured values:** We report results only for N up to 613,889 and k up to 100. Behavior at larger scales is expected to follow the asymptotic trends but has not been measured.
- **Model sizes:** The AI pipeline demonstrations evaluate Qwen3-0.6B and Qwen3-1.7B+LoRA. Larger models (7B, 70B) were not benchmarked due to single-GPU memory constraints. However, SparseDSP's stage speedup is model-independent (Table II reports stage-only times with no model involvement). Larger models increase $\text{TTFT}_{\text{model}}$, which reduces E2E speedup ratio per Equation 5 but does not affect absolute time savings. For a hypothetical 7B model with $\text{TTFT}_{\text{model}} = 500$ ms, the same 32 ms stage saving from $O(\sqrt{N}\log k)$ yields $S_{\text{E2E}} = 550/518 = 1.06\times$, smaller in ratio but identical in absolute savings.
- **Preprocessing not timed:** The benchmarks time discovery from the point where the relevance signal array $x[n]$ is available. The cost of computing $x[n]$ (embedding similarity, BM25 scoring, or other relevance estimation) is not included in stage timings. In production pipelines, preprocessing cost varies by relevance function and may dominate total latency for some workloads. End-to-end pipeline profiling including preprocessing is deferred to future work.

IX. Conclusion

Dense discovery is a fundamental bottleneck in large-scale data processing pipelines. SparseDSP addresses it with regime-adaptive routing among dense scan, the $O(k \log k)$ fast path, and the $O(\sqrt{N} \log k)$ broad discovery path, each deterministic and exact under the (k, Δ)-recoverable model.

On C4 at N=613,889 chunks with k=10, the $O(k \log k)$ algorithm achieves $5.5\times$ speedup over dense scan and $49.6\times$ over dense FFT, translating to $1.38\times$ end-to-end TTFT with an unchanged Qwen3-0.6B model. The $O(\sqrt{N} \log k)$ algorithm's sublinear scaling crosses below dense scan at projected $N \approx 6.5$M. On GSM8K (N=2,795), dense scan wins, validating regime-aware fallback. All configurations achieve 100% accuracy and determinism.

These results suggest that discovery warrants dedicated optimization as a distinct pipeline stage. Unlike domain-specific kernel improvements, which are architecture-bound and subject to diminishing returns, discovery acceleration is domain-agnostic and scales with data growth. As data stores continue to grow faster than processing architectures evolve, the relative importance of pipeline discovery will increase.

A. Future Work

Several directions remain for future investigation:

- **Amortized discovery:** Incremental refinement across sequential queries, reusing partial results from prior discovery calls to reduce per-call cost.
- **Training data selection:** Applying SparseDSP to training pipelines for curriculum selection, hard-negative mining, and data deduplication.
- **Hardware offload:** Exploring memory-compute architectures for in-controller discovery, building on prior work in sparse FFT for memory-compute systems [9].

Acknowledgments

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 an ongoing SparseTech research initiative on deterministic sublinear discovery algorithms for large-scale data processing.

Certain methods and systems described in this work are the subject of a pending patent application.

References

[1] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 16 344–16 359.

[2] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," *arXiv preprint arXiv:2307.08691*, 2023.

[3] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," in *arXiv preprint arXiv:2004.05150*, 2020.

[4] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big Bird: Transformers for longer sequences," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 17 283–17 297.

[5] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. Wang, and B. Chen, "H2O: Heavy-hitter oracle for efficient generative inference of large language models," *arXiv preprint arXiv:2306.14048*, 2023.

[6] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen, "SnapKV: LLM knows what you are looking for before generation," *arXiv preprint arXiv:2404.14469*, 2024.

[7] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[8] A. R. Flouro and S. P. Chadwick, "Keyed gating for multi-view sparse FFT: rigorous guarantees and O($\sqrt{N} \log k$) preprocessing," SparseTech, Tech. Rep., 2025.

[9] ——, "Empirical validation of O($k \log k$) sparse FFT for memory-compute architectures," SparseTech, Tech. Rep., 2024.

[10] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[11] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2017.

[12] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[13] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[14] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2021.

[15] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2020.

[16] 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)*, 2012, pp. 563–578.

[17] ——, "Simple and practical algorithm for sparse Fourier transform," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012, pp. 1183–1194.

[18] S. Pawar and K. Ramchandran, "FFAST: An algorithm for computing an exactly k-sparse DFT in O($k \log k$) time," in *IEEE International Symposium on Information Theory (ISIT)*, 2013, pp. 1324–1328.

[19] M. A. Iwen, "Combinatorial sublinear-time Fourier algorithms," *Foundations of Computational Mathematics*, vol. 10, no. 3, pp. 303–338, 2010.

[20] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan, "Time bounds for selection," *Journal of Computer and System Sciences*, vol. 7, no. 4, pp. 448–461, 1973.

[21] Qwen Team, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.