## Cover Sheet Information (Form PTO/SB/16)

- **Type of Application:** Provisional

- **Title of Invention:** Adaptive Audio Encoding and Compression Toolkit with Tunable Precision (ALAK Toolkit)

- **Inventor(s):** Austin Lakata, 38 W Main St Unit 5, Johnstown, NY 12095-2311, USA (US Citizen)

- **Correspondence Address:** Austin Lakata, 38 W Main St Unit 5, Johnstown, NY 12095-2311, USA

- **Entity Status:** Micro Entity (Individual Inventor qualifying under 35 U.S.C. § 123)

- **Government Interest:** None

- **Related Applications:** This application claims the benefit of priority under 35 U.S.C. § 119(e) from the U.S. Provisional Patent Applications listed below, each of which is incorporated herein by reference in its entirety:

  - Application No. 63/816,434: "Base-i Signal Encoding and Reconstruction System for High-Precision, Low-Noise Signal Processing" (filed June 2, 2025).

  - Application No. 63/832,810: "Base-i and Cube Root Reconstruction Methods for Signal Scrubbing and Encoding" (filed June 30, 2025).

  - Application No. 63/837,447: "Signal Encoding and Reconstruction via Modulo-N Root Systems" (filed July 2, 2025).

  - Application No. 63/843,994: "Mod-N Transform-Based Encoding Method for Digital Signals and Multimedia Data Compression" (filed July 14, 2025).

  - Application No. 63/850,702: "Hybrid Irrational-Prime Modular Phase Ensembles for Noise-Resilient Quantum Signal Reconstruction" (filed July 25, 2025).

## Specification

### Title of the Invention
Adaptive Audio Encoding and Compression Toolkit with Tunable Precision (ALAK Toolkit)

### Cross-Reference to Related Applications
This application claims the benefit of priority under 35 U.S.C. § 119(e) from the U.S. Provisional Patent Applications listed above, each of which is incorporated herein by reference in its entirety. The inventions described herein build upon and extend the modular encoding frameworks (e.g., Base-i, Cube Roots, Mod-N systems) disclosed in those applications, particularly by applying them to audio signal compression with tunable precision and potential integration into quantum-resilient pipelines.

### Statement Regarding Federally Sponsored Research or Development
Not applicable.

### Background of the Invention
1. **Field of the Invention**
The present invention relates to digital signal processing, audio encoding/decoding, data compression, and hybrid quantum-classical computing. Specifically, it discloses a toolkit for adaptive audio compression using tunable precision levels, sparse representation techniques, and extensions to modulations, ensembles, and complex-valued formats for applications in multimedia, telecommunications, and quantum noise mitigation.

2. **Description of Related Art**
Existing audio codecs, such as MP3 (lossy perceptual encoding) and FLAC (lossless compression), provide trade-offs between file size and fidelity but lack fine-grained tunability for diverse signal types, such as test tones, speech, or high-fidelity music. Lossy methods like MP3 achieve ~85-95% compression but introduce artifacts (e.g., muffled highs at low bitrates ~32-128 kbps). Lossless codecs like FLAC offer perfect reconstruction with ~50-60% compression but perform poorly on sparse or repetitive signals.

Recent advancements in neural codecs and quantum-inspired methods (e.g., phase ensembles for noise resilience) show promise, but they are computationally intensive and not optimized for lightweight toolkits. The inventor's prior work in Mod-N encoding frameworks (e.g., Base-i using 4th roots of unity for reconstruction, Cube Roots for three-phase systems, and hybrid irrational-prime ensembles for quantum applications) provides a foundation for improved signal processing. These methods enable low-residue reconstruction, high SNR, and resilience to noise, but have not been fully adapted to audio compression with tunable precision.

There is a need for a modular toolkit that combines time-domain quantization, sparse base-i representation, and adaptive optimizations to achieve extreme compression (e.g., ~99% on silence) while maintaining high fidelity (e.g., ~58 dB SNR at moderate bitrates, rivaling MP3 320 kbps perceptually). The ALAK Toolkit addresses this by extending prior Mod-N inventions to audio, with potential for quantum integrations via complex-valued separation and phase staggering.

### Summary of the Invention
The ALAK Toolkit is a command-line suite for recording, encoding, decoding, compressing, and analyzing audio using a novel adaptive method. Core innovations include:
- Tunable decimal precision (dp) levels (raw/dp1-dp4) for quantization, yielding ~20 dB SNR steps per level.
- Standard mode: Time-domain deltas, zigzag mapping, and RLE for sparsity exploitation.
- Base-i mode (basei_sparse_v1): rotate by the 4-phase cycle [1, i, −1, −i] at level L; quantize; prune; then pack four streams using delta→zigzag→RLE: real_idx_zz_rle, real_val_zz_rle, imag_idx_zz_rle, imag_val_zz_rle along with scale, level, and original_length.
- Auto-heuristics for RLE (run stats) and pipeline selection (spectral variance).
- Pre-processing (normalization, denoising) and post-options (binary MessagePack for ~64% raw size reduction, compressor chaining).
- Extensions to modulations (e.g., irrational/prime phases), ensembles for fusion, and quantum noise mitigation.

Test results demonstrate superior performance: e.g., hi-fi music dp3 at ~344 kbps with transparent quality vs. MP3 320 kbps; vocal dp3 binary.bz2 at 68KB with 55 dB SNR.

The invention is implemented in Python, with potential for hardware acceleration and quantum hybrids.

In development is a novel method for complex SNR in audio reconstruction, specifically for the base-i mode, covering separate evaluation of real and imaginary components to optimize sparsity and phase alignment.

### Detailed Description of the Invention
The ALAK Toolkit, developed by austinLacoustic LLC, is a modular system for audio signal processing. As of August 13, 2025, the toolkit includes CLI modules for recording (cli/record_to_alak.py), encoding (cli/encode_file.py), decoding (cli/decode_file.py), compression (cli/compress_alak_file.py), playback (cli/alak_player.py), signal generation (cli/generate_test_signals.py), and analysis (evaluation/analyze_any.py). Core processing occurs in core/encoder.py and core/decoder.py, with I/O in formats/alak_io.py (supporting JSON/binary MessagePack).

#### 1. System Architecture
- **Input**: Mono float32 PCM from sounddevice (recording) or soundfile (files).
- **Pre-Processing**: Optional normalization to [-1,1] (np.max(abs(signal))); high-pass denoising (Butterworth filter, auto-detect low-energy ratio >5% below 50Hz for cutoff ~20-40Hz).
- **Encoding Pipeline (core/encoder.py)**:
  - User selects dp (raw/dp1-4) and pipeline (AUTO/standard/base-i).

  - For standard (time_intzz_v1): Quantize (scale=10^dp, rint to int32), prune (<prune_counts=0), delta, zigzag, optional RLE if auto (avg_run_len >=4 or zero_ratio >=0.3 or max_run >=16).
  - For base-i (basei_sparse_v1): Treat as complex (imag=0), rotate per index group (level=1 default: multiply by cycle [1, i, -1, -i]), re-quantize real/imag, prune, sparse store non-zero indices/values (optional RLE on values).
  - AUTO: Spectral variance <2.0 on log-FFT triggers base-i; RLE auto per stats.
  - Example Code (Encoder Snippet):
```python
   # Quantize and rotate for base-i (vectorized)
      _, scale = quantize_to_ints(signal, dp)
      L = max(1, level)  # rotation group length
      n = orig_len
      idx = np.arange(n, dtype=np.int64)
      rclass = (idx // L) % 4
      y_r = signal.copy()
      y_i = np.zeros_like(signal, dtype=np.float32)
      m1 = (rclass == 1)  # +i
      m2 = (rclass == 2)  # -1
      m3 = (rclass == 3)  # -i
      y_r[m1] = 0.0; y_i[m1] = signal[m1]  # +i
      y_r[m2] = -signal[m2]  # -1 (imag=0)
      y_r[m3] = 0.0; y_i[m3] = -signal[m3]  # -i
      # Quantize real/imag
      q_r = np.rint(y_r * scale).astype(np.int32)
      q_i = np.rint(y_i * scale).astype(np.int32)
      # Prune and sparse
      th = prune_counts
      if th > 0:
         q_r[np.abs(q_r) < th] = 0
         q_i[np.abs(q_i) < th] = 0
      idx_r = np.nonzero(q_r)[0].astype(np.int64)
      val_r = q_r[idx_r].astype(np.int64)
      idx_i = np.nonzero(q_i)[0].astype(np.int64)
      val_i = q_i[idx_i].astype(np.int64)
      # Pack with delta-zz-rle (always applied in basei_sparse_v1)
      real_idx_zz_rle = _pack_delta_zz_rle(idx_r)
      imag_idx_zz_rle = _pack_delta_zz_rle(idx_i)
      real_val_zz_rle = _pack_delta_zz_rle(val_r)
      imag_val_zz_rle = _pack_delta_zz_rle(val_i)
```
- **Decoding (core/decoder.py)**: Inverse operations; for base-i, unrotate with conjugate cycle, take real part.
  - Example Code (Decoder Snippet):

```python
# Unpack delta-zz-rle for indices/values (basei_sparse_v1)
    idx_r = _unpack_delta_zz_rle(payload["real_idx_zz_rle"])
    idx_i = _unpack_delta_zz_rle(payload["imag_idx_zz_rle"])
    val_r = _unpack_delta_zz_rle(payload["real_val_zz_rle"]).astype(np.int32, copy=False)
    val_i = _unpack_delta_zz_rle(payload["imag_val_zz_rle"]).astype(np.int32, copy=False)
    # Scatter to quantized buffers
    q_r = np.zeros((N,), dtype=np.int32)
    q_i = np.zeros((N,), dtype=np.int32)
    m = min(idx_r.size, val_r.size)
    q_r[idx_r[:m]] = val_r[:m]
    m = min(idx_i.size, val_i.size)
    q_i[idx_i[:m]] = val_i[:m]
    # Dequantize
    y_r = q_r.astype(np.float32) / scale
    y_i = q_i.astype(np.float32) / scale
    # Inverse rotation (vectorized)
    out = np.zeros((N,), dtype=np.float32)
    idx = np.arange(N, dtype=np.int64)
    rclass = (idx // L) % 4
    m0 = (rclass == 0)  # +1 => x = real
    m1 = (rclass == 1)  # +i => x = imag
    m2 = (rclass == 2)  # -1 => x = -real
    m3 = (rclass == 3)  # -i => x = -imag
    out[m0] = y_r[m0]
    out[m1] = y_i[m1]
    out[m2] = -y_r[m2]
    out[m3] = -y_i[m3]
    return out
```
- **Output**: .alak JSON/binary dict with scheme, dp, scale, data (signal_zz/rle or real/imag indices/values), original_length. Binary (MessagePack) reduces raw size ~64% (e.g., 860KB JSON to 308KB bin on 3s vocal).
- **Compression**: bz2/gzip/lzma/zstd chaining; conversions between JSON/binary.
- **Evaluation**: SNR/MSE/MAE, analysis (peak/RMS/centroid/silence ratio); tests show ~20 dB SNR per dp step.

The toolkit extends prior Mod-N inventions by applying base-i/cube roots to audio, with complex separation enabling quantum ties (e.g., phase staggering for NISQ resilience).

#### 2. Test Results and Utility
- Synthetics (sine 220Hz 3s): dp4 SNR ~83 dB, bz2 ~3KB.
- Hi-fi music (10s): dp3 SNR ~58 dB, bz2 ~420KB (~344 kbps), transparent vs. MP3 320 kbps.
- Vocals (3s dp3 binary.bz2): ~68KB, SNR ~55 dB; normalization boosts volume.

- Utility: Archival (raw/dp4 lossless-like), mobile (dp1 ~3-68kbps), quantum (complex separation for phase ensembles).

#### 3. Figures (See Appendix)

### Alternative Embodiments
In some embodiments, the toolkit may incorporate additional modulations (e.g., irrational constants like π, e, φ or primes for phase cycles), ensembles for fusion of multiple Mod-N variants with weighted averaging based on SNR/entropy/deviation metrics, and complex-valued formats for quantum extensions (e.g., real/imag separation for phase staggering in entangled states like GHZ chains, adaptive staggering, echo sequences).

In development is a novel method for complex SNR in audio reconstruction, specifically for the base-i mode, covering separate evaluation of real and imaginary components to optimize sparsity and phase alignment. This could involve computing SNR_real = 10 * log10(P_real_signal / P_real_noise), SNR_imag = similar for imag, then averaging or vectorizing (e.g., total SNR = 10 * log10((P_real + P_imag) / (P_real_noise + P_imag_noise))), enabling fine-tuned optimization of rotation levels and sparsity.

Other potential developments include:
- Binary payload serialization (e.g., MessagePack) for 20-50% raw size reduction.
- Adaptive pruning thresholds based on signal percentiles for dynamic noise reduction.
- Multi-level RLE for nested repetitions, potentially compounding 10-30% savings.
- Custom compressor chaining (e.g., RLE → bz2 → zstd) with entropy-guided selection.
- Entropy-guided hybrid domain switching (time/frequency per block) for 20-40% better compression on music.
- AI-powered pruning/prediction (e.g., LSTM for delta errors) for 15-30% gains.
- Dynamic scale adaptation per block for variable precision, reducing bitrates 20-30%.
- GPU-parallel encoding (CUDA/torch) for 10x speed in batch/multi-channel.
- Generative recovery (diffusion models to upscale low-dp artifacts) for 20-30% perceived quality boost.

- Integrated GUI previews (pygame for dp sliders with SNR estimates) for seamless UX.
- Sustainable low-power mode (SIMD/downsampling low-energy blocks) for 30-50% energy savings.
- Multimodal extension (A/V sync with shared RLE) for 30% media savings in AR/VR.
- Accessibility enhancements (mid-freq dp boost for speech) for 20-30% better intelligibility.
- Wavelet hybrid (pywavelets for highs) for 20-50% size cuts at same SNR.

These embodiments may be combined with prior inventions for hybrid quantum-classical pipelines, such as "energy-aligned" phasing using natural constants.

### Potential Claims
Although not required for provisional applications, the following outline potential utility claims:
1. A method for audio encoding comprising tunable decimal precision quantization, delta encoding, zigzag mapping, and optional run-length encoding for sparse signals.
2. The method of claim 1, further comprising base-i sparse reconstruction with rotational phasing on complex representations, storing non-zero real/imag indices/values.
3. The method of claim 2, extended to modulations using irrational/transcendental or prime-based phases for cycle generation.
4. A system for hybrid quantum-classical signal processing, using the method of claim 3 with ensembles for fusion of multiple encodings, weighted by metrics like deviation, entropy, SNR, to mitigate NISQ noise in audio reconstruction.
5. The system of claim 4, incorporating complex/real separation for phase staggering in entangled states, enabling applications in quantum sensing and error correction.
6. A method for complex SNR evaluation in audio reconstruction, comprising separate computation of SNR for real and imaginary components to optimize sparsity and phase alignment in base-i encoding.

### Appendix - Figures
- Figure 1: Encoding Flowchart
- Figure 2: Standard Encoding Flowchart
- Figure 3: Base-i Encoding Flowchart

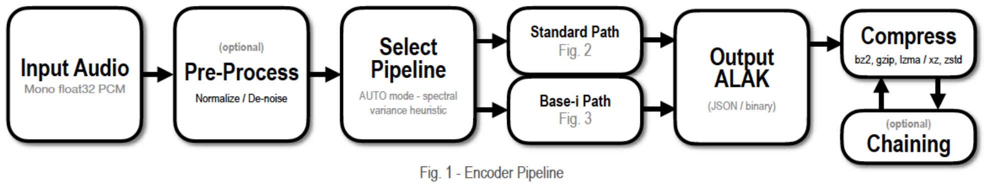


Fig. 1 - Encoder Pipeline

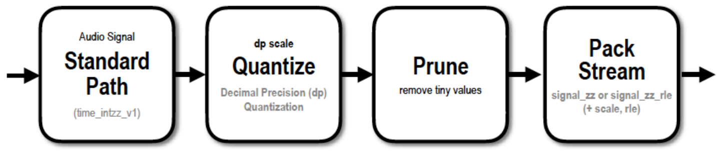


Fig. 2 - Standard Encoding Path

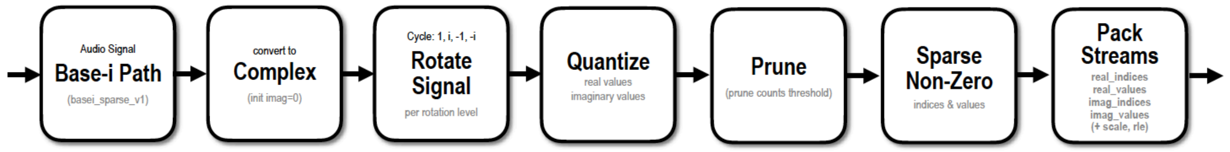


Fig. 3 - Base-i Encoding Path

- Figure 4: Decoding Flowchart
- Figure 5: Standard Decoding Flowchart
- Figure 6: Base-i Decoding Flowchart

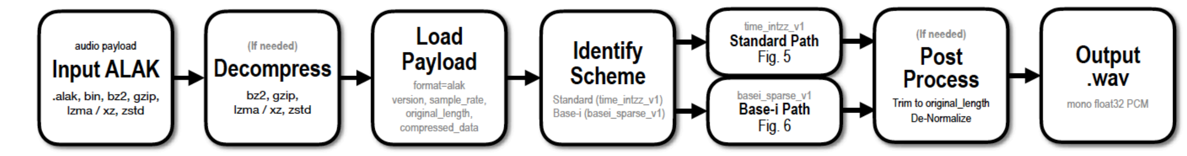


Fig. 4 - Decoder Pipeline

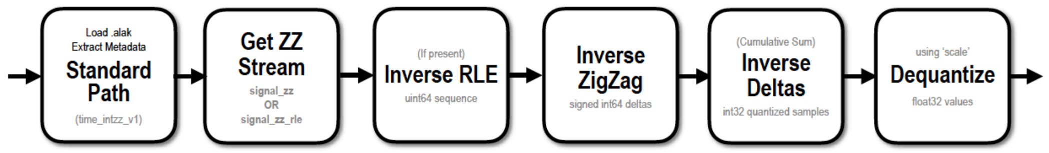


Fig. 5 - Standard Decoder Path

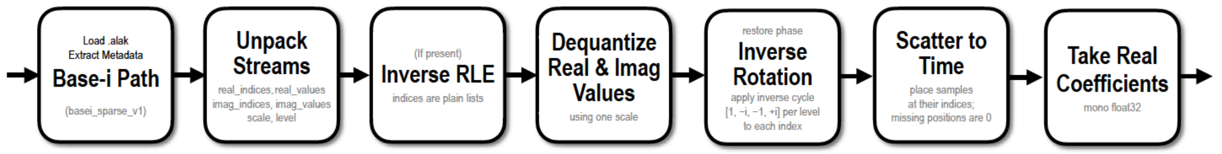


Fig. 6 - Base-i Decoder Path

- Figure 7: dp Auto Selection Process

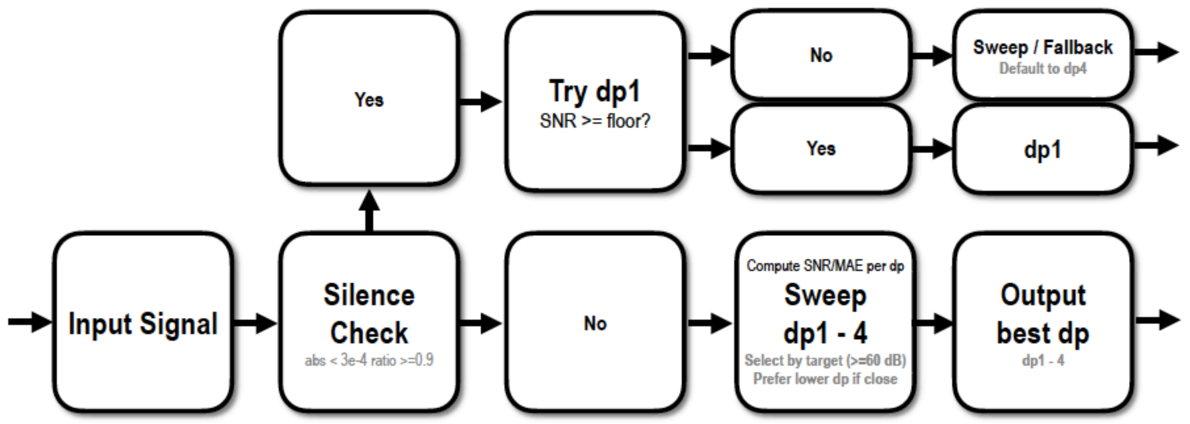


Fig. 7 - dp Auto Selection Process

- Figure 8: RLE Auto Decision Process

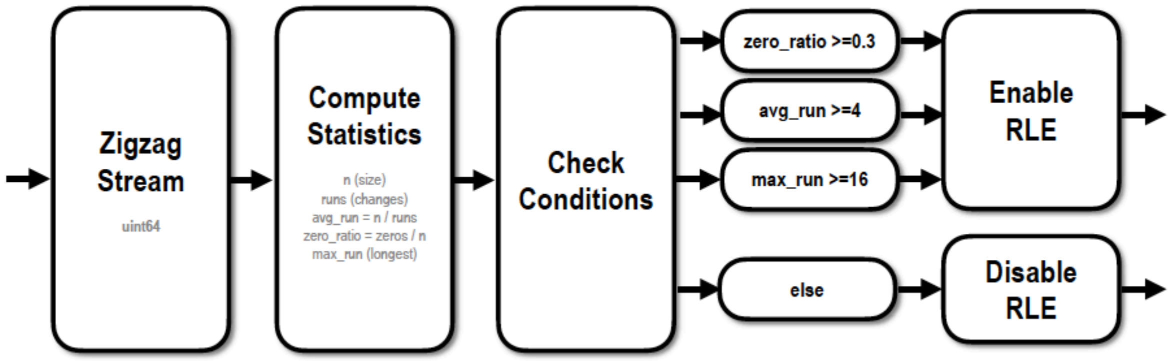


Fig. 8 - RLE Auto Decision Process

- Figure 9: Complex SNR Computation

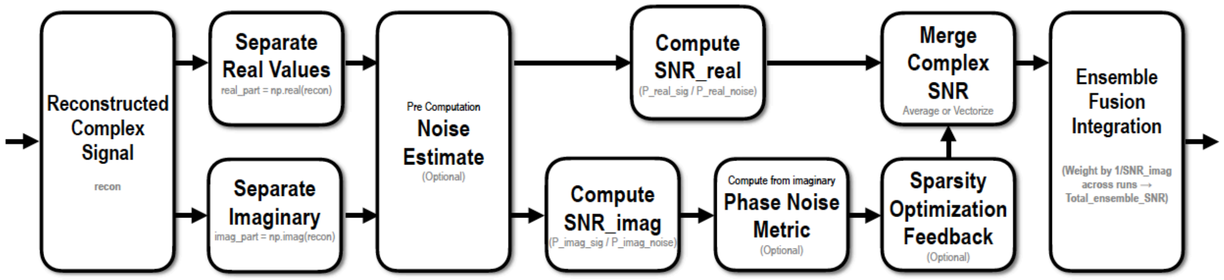
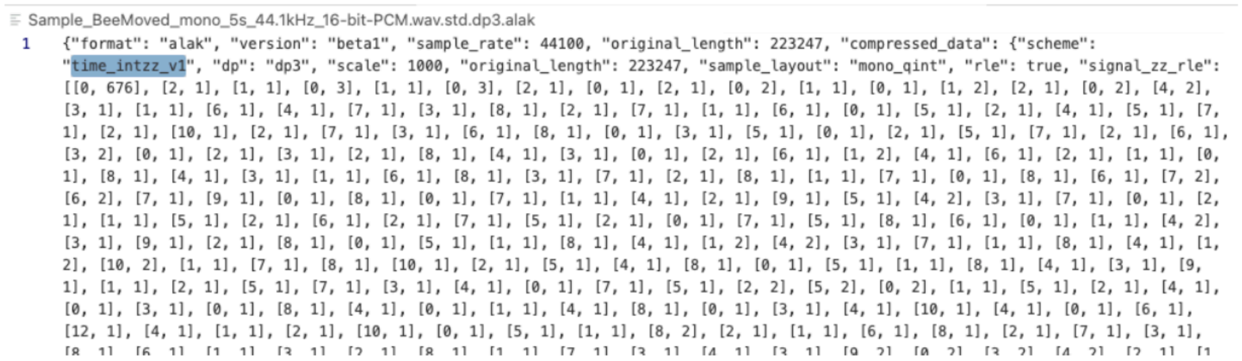

Reconstructed Complex Signal
recon
Separate Real Values
real_part = np.real(recon)
Separate Imaginary
imag_part = np.imag(recon)
Pre Computation
Noise Estimate
(Optional)
Compute SNR_real
(P_real_sig / P_real_noise)
Compute SNR_imag
(P_imag_sig / P_imag_noise)
Compute from imaginary
Phase Noise Metric
(Optional)
Merge Complex SNR
Average or Vectorize
Sparsity Optimization Feedback
(Optional)
Ensemble Fusion Integration
(Weight by 1/SNR_imag across runs → Total_ensemble_SNR)

Fig. 9 - Complex SNR Computation

- Figure 10: Standard Container JSON Scheme (time_intzz_v1)

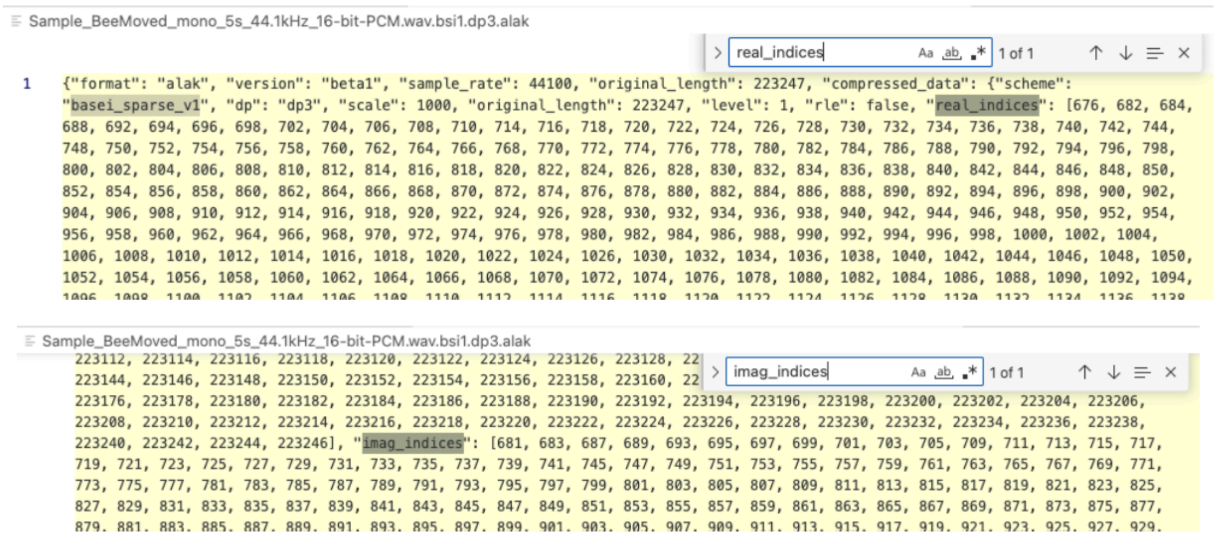


Sample_BeeMoved_mono_5s_44.1kHz_16-bit-PCM.wav.std.dp3.alak

```
{"format": "alak", "version": "beta1", "sample_rate": 44100, "original_length": 223247, "compressed_data": {"scheme":
"time_intzz_v1", "dp": "dp3", "scale": 1000, "original_length": 223247, "sample_layout": "mono_qint", "rle": true, "signal_zz_rle":
[[0, 676], [2, 1], [1, 1], [0, 3], [1, 1], [0, 3], [2, 1], [0, 1], [2, 1], [0, 2], [1, 1], [0, 1], [1, 2], [2, 1], [0, 2], [4, 2],
[3, 1], [1, 1], [6, 1], [4, 1], [7, 1], [3, 1], [8, 1], [2, 1], [7, 1], [1, 1], [6, 1], [0, 1], [5, 1], [2, 1], [4, 1], [5, 1], [7,
1], [2, 1], [10, 1], [2, 1], [7, 1], [3, 1], [6, 1], [8, 1], [0, 1], [3, 1], [5, 1], [0, 1], [2, 1], [5, 1], [7, 1], [2, 1], [6, 1],
[3, 2], [0, 1], [2, 1], [3, 1], [2, 1], [8, 1], [4, 1], [3, 1], [0, 1], [2, 1], [6, 1], [1, 2], [4, 1], [6, 1], [2, 1], [1, 1], [0,
1], [8, 1], [4, 1], [3, 1], [1, 1], [6, 1], [8, 1], [3, 1], [7, 1], [2, 1], [8, 1], [1, 1], [7, 1], [0, 1], [8, 1], [6, 1], [7, 2],
[6, 2], [7, 1], [9, 1], [0, 1], [8, 1], [0, 1], [7, 1], [1, 1], [4, 1], [2, 1], [9, 1], [5, 1], [4, 2], [3, 1], [7, 1], [0, 1], [2,
1], [1, 1], [5, 1], [2, 1], [6, 1], [2, 1], [7, 1], [5, 1], [2, 1], [0, 1], [7, 1], [5, 1], [8, 1], [6, 1], [0, 1], [1, 1], [4, 2],
[3, 1], [9, 1], [2, 1], [8, 1], [0, 1], [5, 1], [1, 1], [8, 1], [4, 1], [1, 2], [4, 2], [3, 1], [7, 1], [1, 1], [8, 1], [4, 1], [1,
2], [10, 2], [1, 1], [7, 1], [8, 1], [10, 1], [2, 1], [5, 1], [4, 1], [8, 1], [0, 1], [5, 1], [1, 1], [8, 1], [4, 1], [3, 1], [9,
1], [1, 1], [2, 1], [5, 1], [7, 1], [3, 1], [4, 1], [0, 1], [7, 1], [5, 1], [2, 2], [5, 2], [0, 2], [1, 1], [5, 1], [2, 1], [4, 1],
[0, 1], [3, 1], [0, 1], [8, 1], [4, 1], [0, 1], [1, 1], [4, 1], [8, 1], [0, 1], [3, 1], [4, 1], [10, 1], [4, 1], [0, 1], [6, 1],
[12, 1], [4, 1], [1, 1], [2, 1], [10, 1], [0, 1], [5, 1], [1, 1], [8, 2], [2, 1], [1, 1], [6, 1], [8, 1], [2, 1], [7, 1], [3, 1],
[8, 1], [6, 1], [1, 1], [3, 1], [2, 1], [8, 1], [1, 1], [7, 1], [3, 1], [4, 1], [3, 1], [9, 2], [0, 2], [3, 2], [4, 2], [2, 1], [1
```

Fig. 10 - Standard Container JSON Scheme (time_intzz_v1)

- Figure 11: Base-i Container JSON Scheme (basei_sparse_v1)



Sample_BeeMoved_mono_5s_44.1kHz_16-bit-PCM.wav.bsi1.dp3.alak

```
{"format": "alak", "version": "beta1", "sample_rate": 44100, "original_length": 223247, "compressed_data": {"scheme":
"basei_sparse_v1", "dp": "dp3", "scale": 1000, "original_length": 223247, "level": 1, "rle": false, "real_indices": [676, 682, 684,
688, 692, 694, 696, 698, 702, 704, 706, 708, 710, 714, 716, 718, 720, 722, 724, 726, 728, 730, 732, 734, 736, 738, 740, 742, 744,
748, 750, 752, 754, 756, 758, 760, 762, 764, 766, 770, 772, 774, 776, 778, 780, 782, 784, 786, 788, 790, 792, 794, 796, 798,
800, 802, 804, 806, 808, 810, 812, 814, 816, 818, 820, 822, 824, 826, 828, 830, 832, 834, 836, 838, 840, 842, 844, 846, 848, 850,
852, 854, 856, 858, 860, 862, 864, 866, 868, 870, 872, 874, 876, 878, 880, 882, 884, 886, 888, 890, 892, 894, 896, 898, 900, 902,
904, 906, 908, 910, 912, 914, 916, 918, 920, 922, 924, 926, 928, 930, 932, 934, 936, 938, 940, 942, 944, 946, 948, 950, 952, 954,
956, 958, 960, 962, 964, 966, 968, 970, 972, 974, 976, 978, 980, 982, 984, 986, 988, 990, 992, 994, 996, 998, 1000, 1002, 1004,
1006, 1008, 1010, 1012, 1014, 1016, 1018, 1020, 1022, 1024, 1026, 1030, 1032, 1034, 1036, 1038, 1040, 1042, 1044, 1046, 1048, 1050,
1052, 1054, 1056, 1058, 1060, 1062, 1064, 1066, 1068, 1070, 1072, 1074, 1076, 1078, 1080, 1082, 1084, 1086, 1088, 1090, 1092, 1094,
1096, 1098, 1100, 1102, 1104, 1106, 1108, 1110, 1112, 1114, 1116, 1118, 1120, 1122, 1124, 1126, 1128, 1130, 1132, 1134, 1136, 1138
```

Sample_BeeMoved_mono_5s_44.1kHz_16-bit-PCM.wav.bsi1.dp3.alak

```
223112, 223114, 223116, 223118, 223120, 223122, 223124, 223126, 223128, 22
223144, 223146, 223148, 223150, 223152, 223154, 223156, 223158, 223160, 22
223176, 223178, 223180, 223182, 223184, 223186, 223188, 223190, 223192, 223194, 223196, 223198, 223200, 223202, 223204, 223206,
223208, 223210, 223212, 223214, 223216, 223218, 223220, 223222, 223224, 223226, 223228, 223230, 223232, 223234, 223236, 223238,
223240, 223242, 223244, 223246], "imag_indices": [681, 683, 687, 689, 693, 695, 697, 699, 701, 703, 705, 709, 711, 713, 715, 717,
719, 721, 723, 725, 727, 729, 731, 733, 735, 737, 739, 741, 745, 747, 749, 751, 753, 755, 757, 759, 761, 763, 765, 767, 769, 771,
773, 775, 777, 781, 783, 785, 787, 789, 791, 793, 795, 797, 799, 801, 803, 805, 807, 809, 811, 813, 815, 817, 819, 821, 823, 825,
827, 829, 831, 833, 835, 837, 839, 841, 843, 845, 847, 849, 851, 853, 855, 857, 859, 861, 863, 865, 867, 869, 871, 873, 875, 877,
879, 881, 883, 885, 887, 889, 891, 893, 895, 897, 899, 901, 903, 905, 907, 909, 911, 913, 915, 917, 919, 921, 923, 925, 927, 929,
```

Fig. 11 - Base-i Container JSON Scheme (basei_sparse_v1) real and imaginary indices

- Figure 12: Comprehensive Test Summary - Standard Pipeline (time_intzz_v1)

```markdown
# test_results/005_TEST_SUMMARY.md
# ALAK Toolkit Beta — Test Summary

---

# ALAK Toolkit Beta — Test Summary

This page ties together deterministic signal tests, broadband noise tests, room-tone
recordings, and compression snapshots. It references detailed logs in:

* `001_TEST_RESULTS.md` (metrics tables)
* `002_TEST_RESULTS.md` (compression outputs)
* `003_TEST_RESULTS.md` (hi-fi music recording)
* `004_TEST_RESULTS.md` (live music recording)

---

## Environment

* Machine: Apple MacBook Pro (M1)
* OS: 13.7.6 (22H625)
* Python: 3.11.9
* Key libs: numpy, scipy, soundfile/libsndfile, zstandard, bz2/lzma/gzip
* ALAK version: beta
* Pipeline: (time_intzz_v1)

## Signals covered

Sine 1 kHz • Chirp 100→5000 Hz • Triangle 1 kHz • Square 250 Hz (50% on-grid ±1.0) •
Square 250 Hz (25% off-grid, amp=0.90123) • Pink noise • Digital silence • Room tone
(recorded) • Music clips

## Method (summary)

1. Generate or record audio via 'main.py' menus.
2. Encode to RAW and dp4/dp3/dp2/dp1 ('.alak').
3. Compare against WAV (WAV as reference) using *Analyze / Compare*.
4. Optionally compress '.alak` via *[5] Compress* (bz2 by default).
5. Report SNR/MAE and file sizes.
```

## Key results (metrics)

* **Decimal precision ladder (dp1→dp4):** On broadband/complex signals, SNR rises by roughly \~20 dB per dp step, matching design expectations observed throughout the tests.
* **Square (on-grid, ±1.0, 50% duty):** RAW ≡ dp1–dp4 (bit-perfect w.r.t. RAW); \~90 dB vs the 16-bit WAV baseline is consistent with PCM rounding.
* **Square (off-grid, amp=0.90123):** dp4 tracks the WAV closely; dp1/dp2 quantize to plateaus (e.g., 0.90123→0.90000), increasing MAE as expected.
* **Pink noise:** The "ladder" is visible (\~16.5/36.6/56.6/76.6 dB for dp1→dp4 in these runs).
* **Digital silence:** Exact equality across RAW and all dp modes (MSE=MAE=0).
* **Room tone:** SNR increases with input gain; at very low levels, coarse dp modes are dominated by quantization—this informed `dp_auto` behavior.

## `dp_auto` behavior (observed chronology)

* 100% input (room tone): **dp4**
* 60% input (room tone): **dp4** (example take measured \~32.1 dB SNR)
* 50% input (room tone): **dp1** (no usable signal detected on that take)
* 50% input (breathing present): **dp4** (\~42.6 dB SNR on that take)
* 40% input (breathing present): **dp1**

> Notes: Early runs prior to the gate tweak could classify \~70% input as near-silence; subsequent tests used the updated gating/logic. The list above reflects the corrected state.

## Compression snapshot (bz2 wrapper)

Exact byte sizes are in `002_TEST_RESULTS.md`. Representative examples (original `.alak` → `.alak.bz2`):

* **Chirp 5 s RAW:** 4,549,898 → 1,462,700 (≈32%)
* **Chirp 5 s dp1:** 685,416 → 17,700 (≈3%)
* **Pink 3 s RAW:** 2,810,180 → 1,041,205 (≈37%)
* **Sine 3 s dp1:** 397,122 → 467 (near-zero)

```markdown
* **Square 3 s (50% on-grid) RAW:** 727,836 → 272 (near-zero)
* **Square 3 s (25% off-grid, amp=0.90123) RAW:** 2,613,145 → 530 (near-zero)

**Interpretation:**
Highly structured or plateau-heavy streams (e.g., silence, on-grid square, coarse
quantized lows) become extremely compressible after delta→zigzag→RLE. Broadband
content compresses less aggressively; dp choice and wrapper matter more.

**Reproduce:**
Use `main.py` → *\[6] Generate test signals* or *\[1] Record audio* → *\[4] Analyze /
Compare* (WAV as reference) → *\[5] Compress*. Keep WAV subtype consistent (`PCM_16`
for PCM baselines or `FLOAT` for float round-trips).

---

## Hi-Fi music clip (10 s mono @ 44.1 kHz, 16-bit)

Source: “Bee Moved” demo. Objective results below are from the toolkit’s in-memory
decode check (WAV → encode → decode vs WAV). Sizes are `.alak.bz2`.

| Mode | SNR (dB) |      MAE | `.alak.bz2` size | ≈Bitrate\* |
| ---- | -------: | -------: | ---------------: | ---------: |
| RAW  |   107.99 | 0.000000 |          1.15 MB | \~965 kbps |
| dp4  |    78.78 | 0.000025 |        621.02 KB | \~509 kbps |
| dp3  |    58.78 | 0.000250 |        420.42 KB | \~344 kbps |
| dp2  |    38.80 | 0.002491 |        217.95 KB | \~178 kbps |
| dp1  |    18.80 | 0.024927 |         82.86 KB |  \~68 kbps |

\*Bitrate = `size_bytes*8 / 10 s`. Kibibyte units (1024) used for sizes.

**Listening notes (single listener, casual):**

* RAW: very high fidelity; low-level details audible relative to the 16-bit PCM
baseline.
* dp4: transparent in casual listening on this clip.
* dp3: perceived as essentially transparent on this clip.
* dp2: very good; competitive subjectively at a substantially smaller size than
high-rate encodes in this test.
* dp1: audible wideband hiss/“static,” but high-frequency cues remained present on
this material.

> These are clip-specific and informal; no formal ABX was performed.
```

## Live music clip (10 s mono @ 44.1 kHz, 16-bit)

Source: Grateful Dead — “Lazy River Road” (1995-03-17). Round-trip quality (WAV → ALAK → decode vs WAV):

Mode	SNR (dB)	MAE
RAW	83.90	0.000000
dp4	54.68	0.000025
dp3	34.68	0.000250
dp2	14.69	0.002498
dp1	0.41	0.011198

Compression (bz2) for this low-level program (9.98 s):

Mode	`.alak` size	`.alak.bz2` size	≈Bitrate (kbps)
RAW	7.76 MB	**783.71 KB**	**\~643**
dp4	1.83 MB	**496.24 KB**	**\~407**
dp3	1.44 MB	**291.63 KB**	**\~239**
dp2	1.26 MB	**109.48 KB**	**\~90**
dp1	1.26 MB	**4.29 KB**	**\~3.5**

Why the extreme dp1/dp2 reduction here:
The clip’s instantaneous amplitudes are often very small; coarse quantization produces long runs of identical values. The delta→zigzag→RLE chain, followed by bz2, compresses such runs extremely well.

## Notes on comparability

* SNR/MAE are time-domain metrics vs the WAV baseline. They are not psychoacoustic scores.
* Bitrates shown are for `.alak.bz2` as produced by the toolkit (container overhead included) and for mono 44.1 kHz clips of stated duration.
* Subjective comments are single-listener and clip-specific.

## Conclusion

Across deterministic, broadband, and recorded material:

* The **dp ladder** behaves as designed, offering predictable quality steps (\~20 dB SNR per dp on complex material).
* **RAW/dp4** provide very high fidelity on all tested material.
* **dp2/dp1** trade fidelity for size in a controlled, content-dependent way; highly structured or low-level material can compress to extremely small payloads due to sparsity after quantization.
* **`dp_auto`** selects conservative precision when content is present and de-escalates to coarse precision on near-silence, matching the observed gating logic.

This summary is strictly based on the measurements and listening notes produced with the ALAK beta toolkit in the environment listed above.
