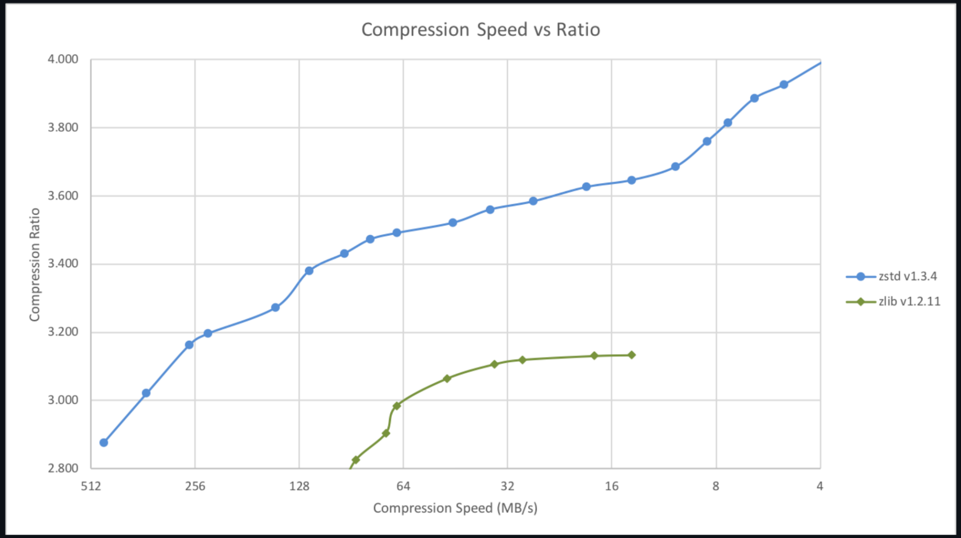
Beyond Zstandard

New directions for lossless data compression

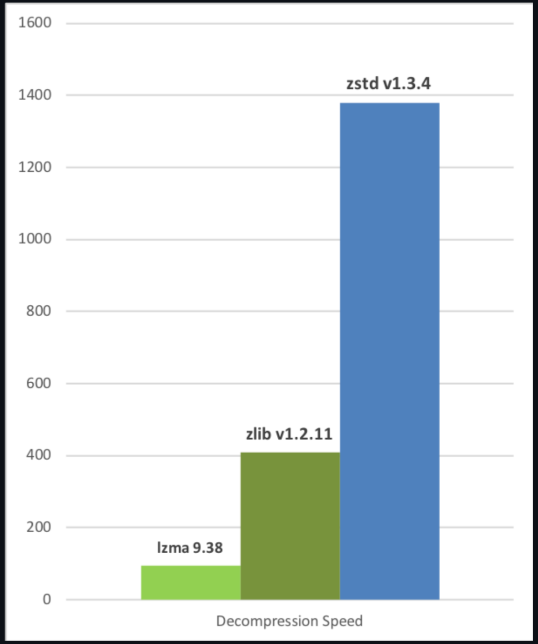
Stanford lecture, EE274, Oct. 2025

Zstandard introduction (2015)

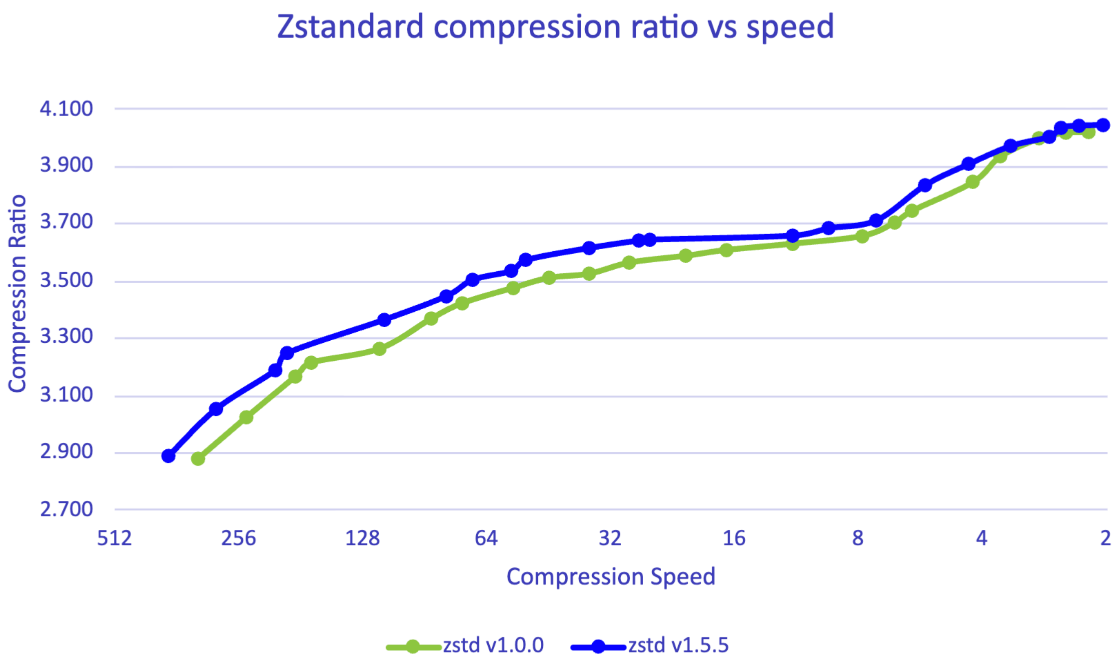
Compression Speed vs Ratio

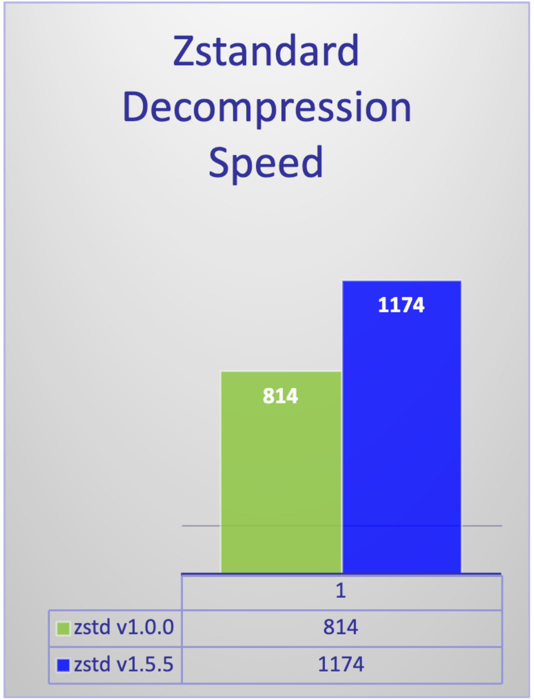


Decompression Speed

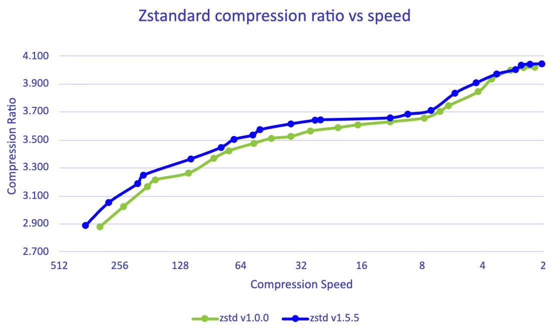


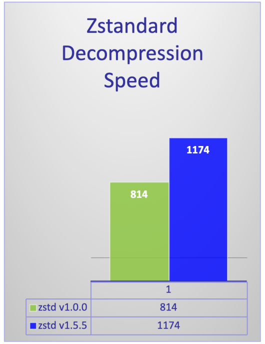
Zstandard evolution





Beyond Zstandard





- Reaching asymptotic limits
  - Small gains for large energy cost
- New LZ77 format?
  - Some small improvement
  - But ecosystem cost: confusion
    - Conjecture: new entrant must offer **significant** improvements
- Other variants (LZ78, ROLZ, Grammar, Repair, etc.)
  - Converge towards same limit
  - Fundamental assumption: data is a bunch of (undifferentiated) bytes
- High compression algorithms (PPM, BWT, CM, NN, etc.)
  - Too slow for datacenters
- Format-specific Compression

A Trivial Example

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

LZ cannot compress this because there are no repetitions

⇩ Delta

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

A simple transform makes it trivially compressible

Conjecture : understanding the data format opens new ways to manipulate it, leading to outsized gains in compression efficiency.

SAO

- Smithsonian Astrophysical Observatory
    - Catalog of stars
- Part of Silesia compression corpus :
    - 7,251,944 bytes
    - 258,997 stars
    - Binary Format

SAO format description

- Header + array of records
- Star record: 28 bytes

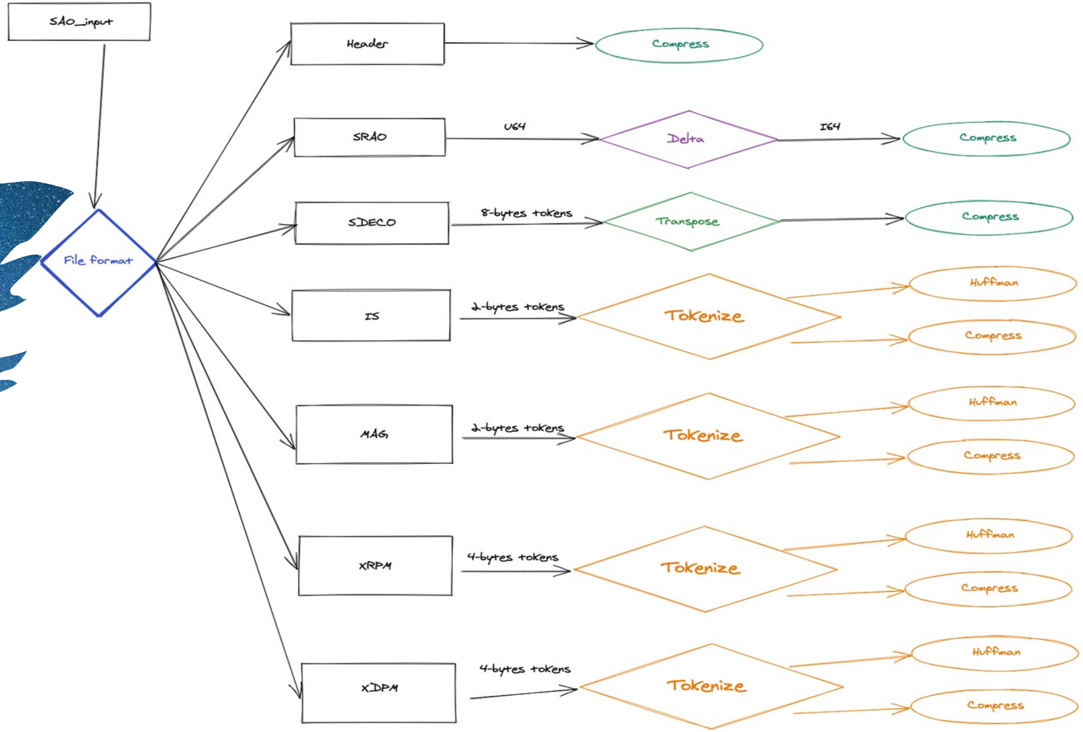
```
Real*8 SRA0
Real*8 SDEC0
Character*2 IS
Integer*2 MAG V
Real*4 XRPM
Real*4 XDPM
```

Coordinates

Attributes

Movements

Example: Simple compressor for SAO



SAO Compression comparison

- Skylake core @3.6 GHz, Ubuntu 24.04, clang-19

	zstd -3			
Compressed Size				
Compression Factor				
Compression Speed				
Decompression Speed				

- Conclusion:
  Exploiting format specification leads to better compression ratio

SAO Compression comparison

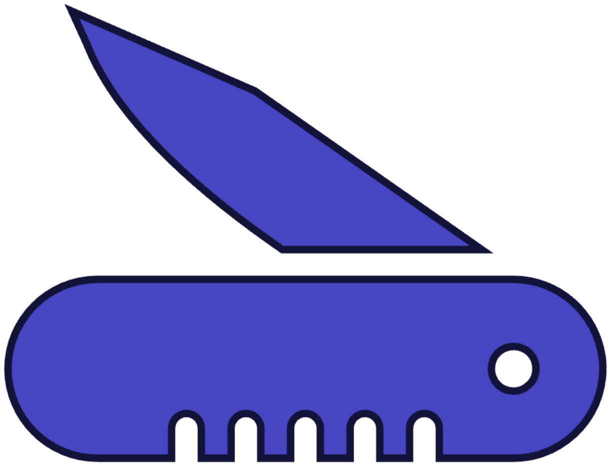
- Skylake core @3.6 GHz, Ubuntu 24.04, clang-19

	zstd -3	lzma -9	cmix	SAO-specific
Compressed Size	5,551,154	4,416,774	3,726,762	3,516,303
Compression Factor	1.31	1.64	1.94	2.06
Compression Speed	100 MB/s	2.9 MB/s	0.001 MB/s	215 MB/s
Decompression Speed	750 MB/s	45 MB/s	0.001 MB/s	800 MB/s

- Conclusion:
  Exploiting format specification leads to better compression ratio
  and better speed

The double edge of format-centric compression

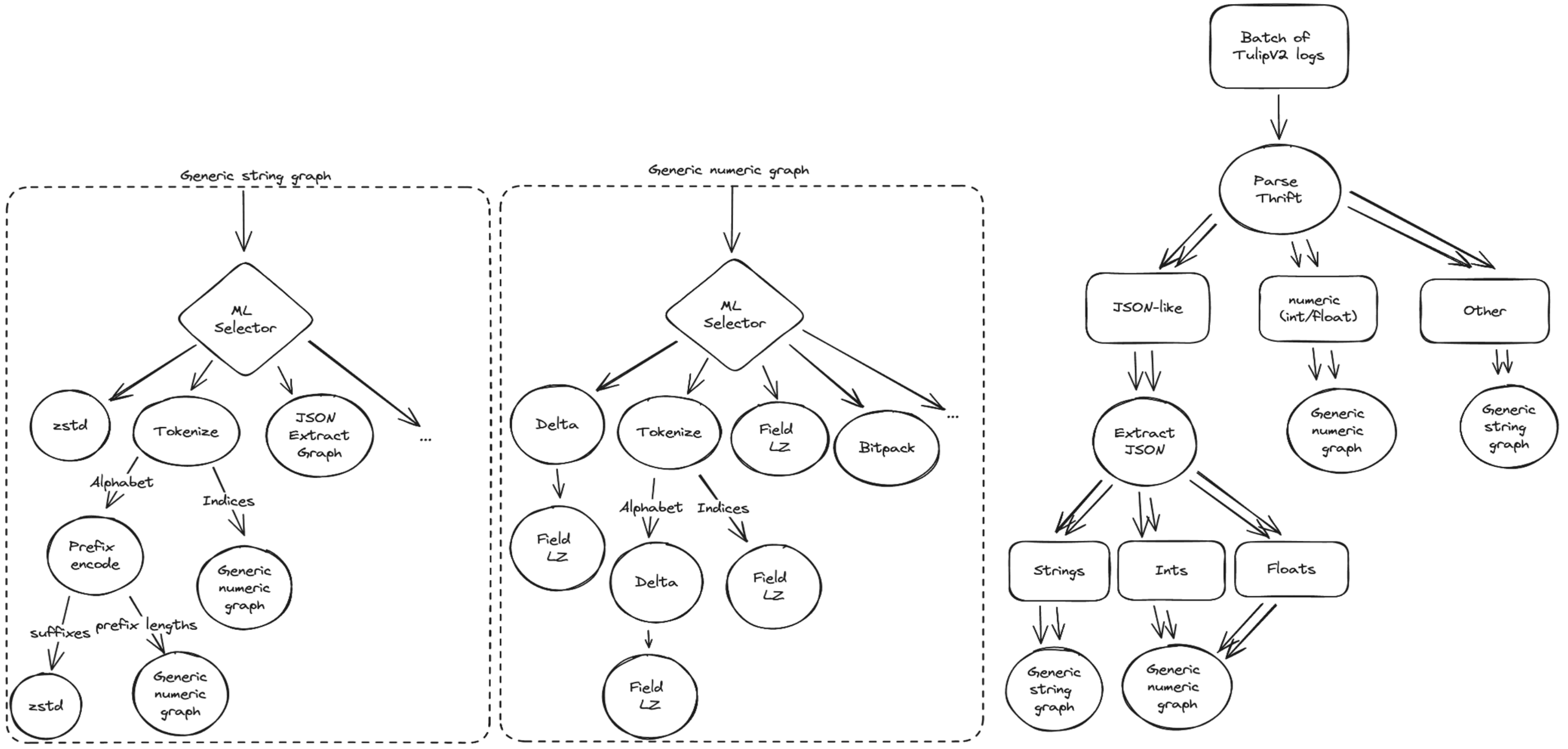
- Time to design
  - Time to learn fundamentals
  - Time and risks to discover a good solution
- Rebuild same fundamental units
  - Time to optimize
  - Time to safeguard (intrusions, fuzzing)
- Tricky deployment and evolutions
  - Decoders must be deployed first across all receivers; only then can the new encoder be employed.
  - Data changes all the time in Datacenters
- Explosive maintenance cost
  - Forgotten code, no one left to support
  - Velocity obstacle, security liability



OpenZL

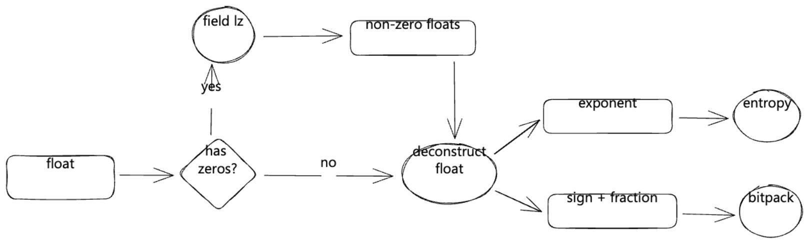
- A core library and suite of tools to generate specialized compressors
  - Compressors as Graphs of pre-validated codecs
  - Explorable solution space, using ML
- Fixing the deployment bottleneck
  - **Unified decompression engine**, valid for any Graph
  - New compressors are just configs — unified decompressor supports both old and new configs simultaneously
  - Compression Graphs can be updated anytime
    - including *during* compression => Dynamicity
- Fixing dead (unmaintained) code headaches
  - Centralized code base => Identified place for maintenance, performance and security

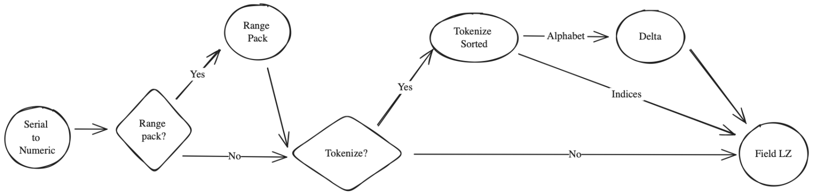
Graph examples (Thrift)



Real-time Graph adjustments

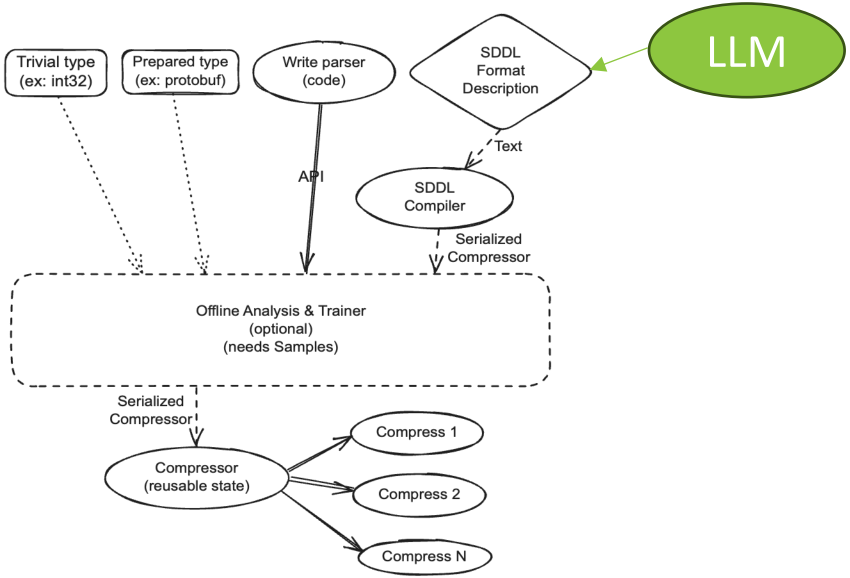
- Graphs can automatically adjust to real input
  - React to natural variability, and exceptional events
  - Trainable
    - Decision can be driven by ML component
  - Selectors
    - Associate a set of features with a direction (follow up Graph) => Classifier





How to generate Compressors

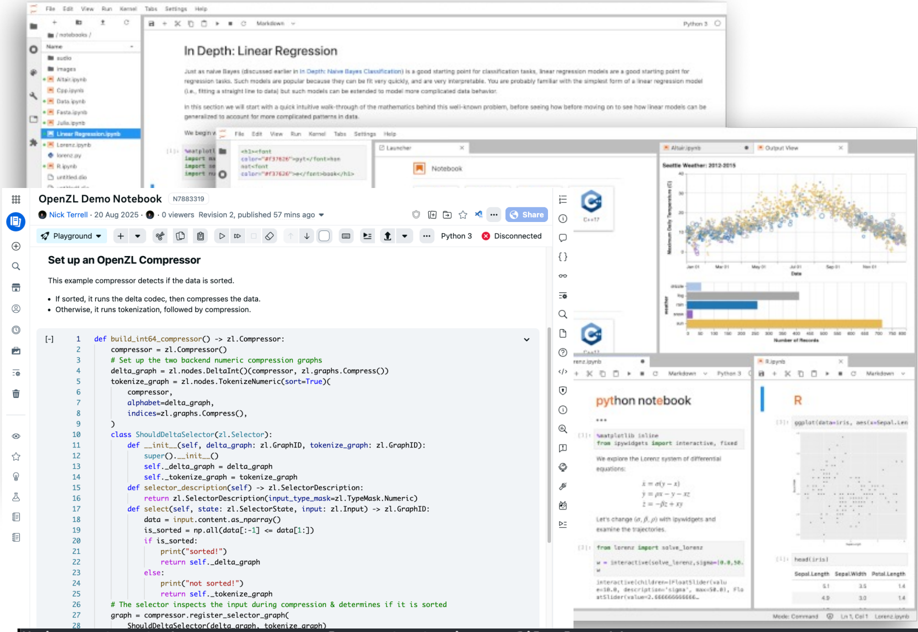
- Describe your data
  - Preset, Parser Function, or SDDL Compiler




```
# SAO - SDDL spec example

# Describe the row structure
StarRecord = {
  SRA0 : UInt64LE   # Right ascension in degrees
  SDEC0: UInt64LE   # Declination in degrees
  IS   : Byte[2]    # Instrument status flags
  MAG  : UInt16LE   # Magnitude * 100
  XRPM :  Int32LE   # X-axis rate per minute
  XDPM :  Int32LE   # X-axis drift per minute
}

# Consume the header
header: Byte[28]

# header is followed by an array of records
data: StarRecord[]
```

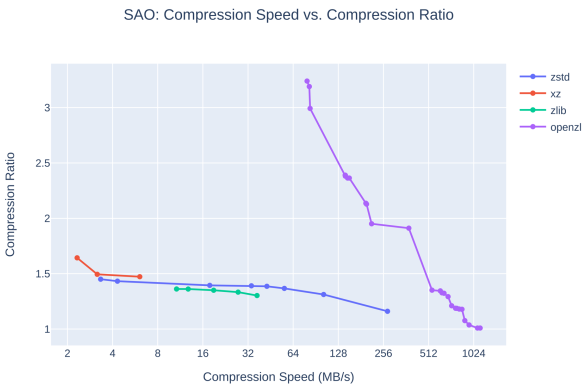
How to generate Compressors

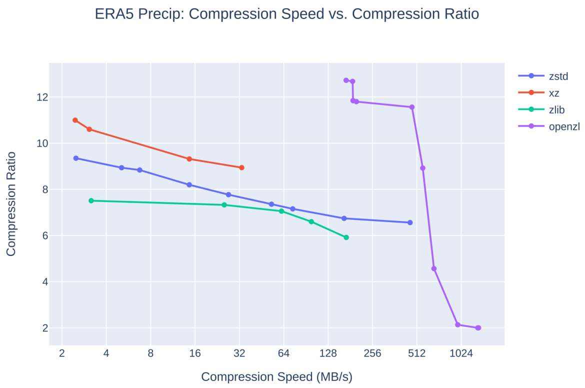
- Describe your data
  - Preset, Parser Function, or generated with SDDL Compiler
- Offline Training
  - Generates a `Compressor Config` from samples + parser
- Manually
  - Tools
    - Exploration, evaluation
  - Inspectable: Debug and Research

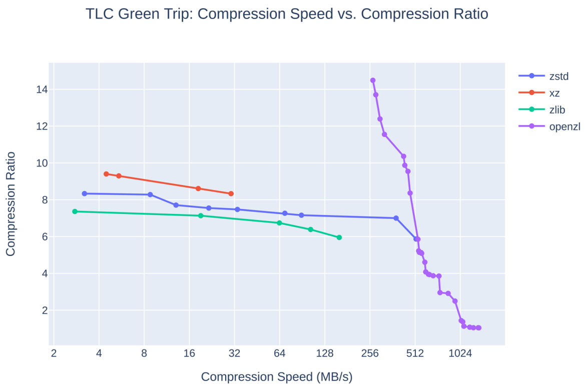


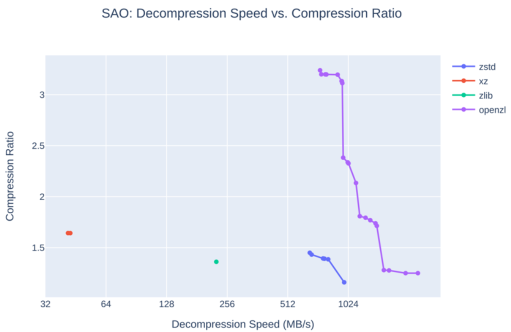

```
Using automated compressor explorer to train profile le-u64
Picked 1 samples out of 1 samples with total size 2071976
Benchmarking untrained compressor...
1 files: 2071976 → 1945607 (1.06),  350.21 MiB/s  854.32 MiB/s
[==================================================] Training ACE graph 1/1
Benchmarking trained compressor...
1 files: 2071976 → 598022 (3.46),  116.19 MiB/s  1034.28 MiB/s
Training improved compression ratio by 225.34%
```

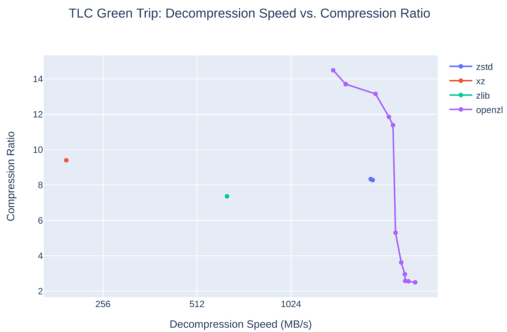
Results

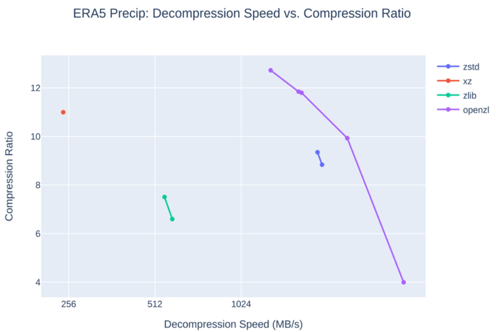

SAO: Compression Speed vs. Compression Ratio
Compression Ratio
1
1.5
2
2.5
3
2
4
8
16
32
64
128
256
512
1024
Compression Speed (MB/s)
zstd
xz
zlib
openzl


ERA5 Precip: Compression Speed vs. Compression Ratio
Compression Ratio
2
4
6
8
10
12
2
4
8
16
32
64
128
256
512
1024
Compression Speed (MB/s)
zstd
xz
zlib
openzl


TLC Green Trip: Compression Speed vs. Compression Ratio
Compression Ratio
2
4
6
8
10
12
14
2
4
8
16
32
64
128
256
512
1024
Compression Speed (MB/s)
zstd
xz
zlib
openzl

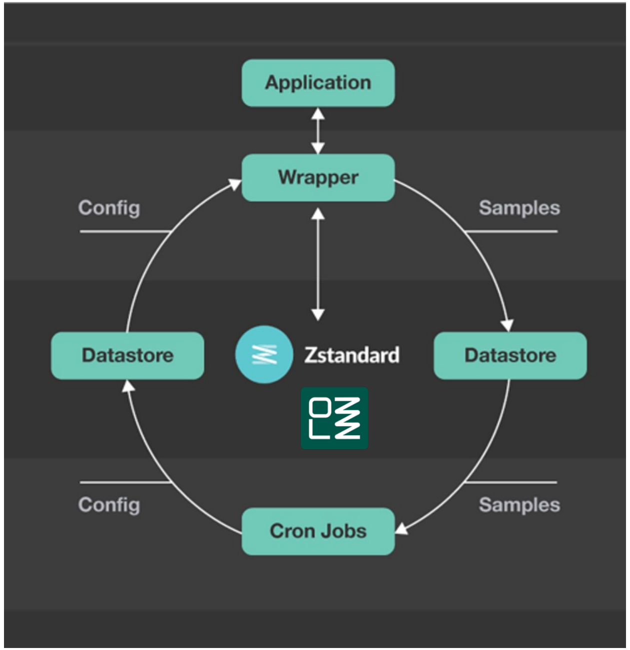

SAO: Decompression Speed vs. Compression Ratio
Compression Ratio
1.5
2
2.5
3
32
64
128
256
512
1024
Decompression Speed (MB/s)
zstd
xz
zlib
openzl


TLC Green Trip: Decompression Speed vs. Compression Ratio
Compression Ratio
2
4
6
8
10
12
14
256
512
1024
Decompression Speed (MB/s)
zstd
xz
zlib
openzl


ERA5 Precip: Decompression Speed vs. Compression Ratio
Compression Ratio
4
6
8
10
12
256
512
1024
Decompression Speed (MB/s)
zstd
xz
zlib
openzl

OpenZL @Meta Infrastructure

- OpenZL is already deployed at scale @Meta
- Primary workload: AI – growing fast
- Better compression => storage savings, faster transmissions => higher compute utilization



Future Directions

- Guess structure from arbitrary binary sample
  - Ex: SDDL as final representation format
- Better inline decisions (control points)
  - Faster, leaner and more powerful ML Selectors
- Playground for novel codecs
  - Reduce requirement to build a lot of infrastructure around

https://openzl.org

Thank You!