nvz — broad evaluation

Empirical evaluation of a content-aware archiver: compression ratio, wall-clock cost, and correctness across eight data types and two standard corpora.

2026-11-16 · Ryzen AI 9 HX 470 (24 threads), 28 GiB RAM, NVMe · source code, measurement data, and this report: <https://github.com/hugowikstrom/nvz> · every figure is generated from `resultat/*.json` and can be reproduced per appendix A · svensk version

---

Abstract

nvz is an archiver that classifies its input and applies content-specific, lossless transforms — deduplication, similarity ordering, PNG inflation with recipe recovery, JPEG re-encoding to arithmetic code, and context-mixing compression (nvcm) — before selecting a codec per block by sampling. This report evaluates nvz against seven established compressors on eleven corpora and tests two hypotheses:

- **H1 (compression):** content awareness yields measurably smaller archives

than generic stream compression on data with exploitable structure, without degrading the result on data that lacks it.

- **H2 (correctness):** every archive reproduces the source tree bit-exactly

— contents, permissions, timestamps, symbolic links — and a damaged archive is rejected rather than yielding incorrect data.

The results support both hypotheses, with one quantified deviation from H1. In summary:

observation	measured	reference
log files, ratio	**23.8×** (10.03 of 238.3 MiB)	xz −9e: 20.1×
log files, vs. best classical	**−15.3 %**	vs. xz −9e
Silesia, nvcm (single thread)	**1.677 bits/byte**	xz −9e: 1.829
enwik8, nvcm (single thread)	**1.713 bits/byte**	xz −9e: 1.986
video, all compressors	within 1 % of raw size	expected: already coded
archives-of-archives, nvz vs. zstd −22	**−0.1 %** after remedy (+20.0 % before)	diagnosed and fixed, section 4.2
restoration, 7 corpora	bit-exact, all entries	section 5
damage tests, 5 damage classes	all rejected	section 5.2

The initially measured deviation from H1 (+20.0 % on the corpus of already-compressed archives) was diagnosed as the codec probe's sample selection, remedied during the evaluation, and re-measured to −0.1 % against zstd −22. A complementary measurement against LZMA2 at full strength (7-Zip, 512 MiB dictionary; section 3.6) moves the reference, however: there nvz trails by +2.3 % on that corpus, while its lead on text rather grows. H1 thus stands with one known, quantified counterexample — archives-of-archives against LZMA2-extreme — and none elsewhere.

---

1. Introduction

A conventional archiving pipeline (`tar | zstd`) treats the tree as an unordered byte stream: the compressor sees neither file duplicates outside its window, nor embedded compression layers (deflate in PNG, Huffman in JPEG), nor the opportunity to place similar files adjacently. nvz exploits all three, and every transform is verified at pack

time by full re-synthesis and comparison against the original; a transform that cannot be proven reversible falls back to raw storage.

Previous measurements covered three homogeneous corpora (source code, PNG, JPEG photographs) and one mixed project tree. They do not answer how the archiver behaves on data without exploitable structure (video, GIF, already-compressed archives), on highly redundant log data, or in comparison with published results. The present evaluation extends the evidence with four corpora of real data, two standard corpora (Silesia, enwik8), and a systematic correctness assessment.

2. Method

2.1 Reference compressors

Seven configurations of four established tools: gzip (−6, −9 via pigz), bzip2 (−9), zstd (−3, −19 `--long=27`, −22 `--ultra --long=31`), and xz (−9e). `tar | zstd −3` is labelled *current* and serves as the baseline, as it matches the machine's existing archival script. nvz is measured in modes `snabb` ("fast": zstd −3, no transforms), `normal` (transforms, codec probe per block), and `liten` ("small": additionally nvcm as a candidate; the default), plus `utan knep` ("no tricks": mode normal with deduplication and transforms disabled) to isolate the transforms' contribution.

2.2 Measurement procedure

Each data point is a complete cycle: pack, unpack into an empty directory, recursive comparison against the source tree (`diff -rq --no-dereference`). The *exact* column in the tables is the outcome of that comparison, not an assumption. Times are wall-clock; packing may use all 24 threads. Measurements ran under `nice 19` and `ionice -c 3` on an otherwise loaded machine with idle cores; control runs showed no measurable effect of the priority. The codec-level measurement (section 3.4) instead follows the field's convention: one file at a time, one thread, no tar, no deduplication.

Archive size is not deterministic across runs: two identical `--läge normal` runs produced 8 597 884 and 8 597 603 bytes (a $3{\cdot}10^{-5}$ difference). Differences within a few hundred bytes are therefore not significant.

2.3 Corpora

The six below complement the earlier three (code 55.3 MiB/5 974 files, PNG 431.1/4 618, photo 408.5/1 143). Local corpora are built as hard links by `korpus/bygg-korpus.py`; the log corpus is instead copied once, since log files are written during measurement and a mutating corpus invalidates the cycle comparison. Files modified within the last 15 minutes are excluded.

corpus	size	contents	rationale
`gif`	105 files, 27.4 MiB	web graphics, animations, duplicates across several `node_modules`	LZW entropy layer that nvz does not open; high duplicate share
`film`	67 files, 536.7 MiB	h264 (TikTok), VP9 (screen recording)	transform-coded data; expected null result
`arkiv`	133 files, 391.8 MiB	zip and tar.gz, incl. two bit-identical pairs and eight daily database backups	compressed data with redundancy *between* files
`logg`	76 files, 238.3 MiB	syslog, kern.log, app logs, JSONL transcripts	maximally repetitive text; a growing archive class
`silesia`	12 files, 211 938 580 B	the Silesia corpus	comparability: standard in lzbench et al.

corpus	size	contents	rationale
`enwik8`	1 file, 10^8 B	English Wikipedia, first 10^8 bytes	comparability: Large Text Compression Benchmark

The duplicate share per corpus is quantified below, since it determines how much of a ratio is explained by deduplication rather than compression.

corpus	files	unique files	raw MiB	exact duplicates MiB	share
kod	5,974	5,898	55.3	0.4	0.7 %
png	4,618	1,963	431.1	123.8	28.7 %
foto	1,143	1,142	408.5	0.0	0.0 %
gif	105	59	27.4	13.0	47.3 %
film	67	67	536.7	0.0	0.0 %
arkiv	133	130	391.8	62.8	16.0 %
logg	76	74	238.3	0.0	0.0 %
silesia	12	12	202.1	0.0	0.0 %
enwik8	1	1	95.4	0.0	0.0 %

3. Results

3.1 Overview

Best nvz mode against the baseline and against the best classical tool per corpus. Negative values denote smaller archives.

corpus	raw MiB	current MiB	best classical MiB	best nvz MiB	vs. current	vs. best klassiska	pack time nvz/classical
Source code	55	11.5	8.2 (xz -9e)	7.2 (liten)	−37.1 %	−12.5 %	68 / 47 s
Images (PNG)	431	362.9	245.2 (zst -22)	207.2 (liten)	−42.9 %	−15.5 %	280 / 156 s
Photos (JPEG)	409	406.3	404.9 (bz2 -9)	345.1 (normal)	−15.1 %	−14.8 %	48 / 57 s
Images (GIF)	27	27.0	14.2 (zst -22)	14.2 (normal)	−47.6 %	−0.0 %	6 / 5 s
Video (h264 and VP9)	537	533.9	532.0 (zst -19)	532.0 (normal)	−0.3 %	+0.0 %	56 / 57 s
Archives (zip and tar.gz)	392	381.8	242.8 (zst -22)	237.0 (liten)	−37.9 %	−2.4 %	323 / 154 s
Loggfiles	238	22.4	11.8 (xz -9e)	10.0 (liten)	−55.2 %	−15.4 %	191 / 897 s
Silesia (archive level)	202	63.1	46.2 (xz -9e)	43.5 (liten)	−31.0 %	−5.7 %	100 / 158 s
enwik8 (archive level)	95	33.8	23.7 (xz -9e)	20.4 (liten)	−39.6 %	−13.7 %	129 / 121 s

3.2 Full matrix

All compressors across all corpora, in three panels: size, pack time, unpack time. The silesia and enwik8 columns refer to the archive level (whole corpus through a tar-equivalent path), not the codec level of 3.4.

Archive size

compressor	kod	png	foto	gif	film	arkiv	logg	silesia	enwik8
tar (okomprimerad)	60.8	438.4	409.4	27.6	536.8	391.9	238.4	202.1	95.4
tar.gz -6	12.3	401.7	405.8	27.2	532.4	380.0	30.5	65.1	34.8
tar.gz -9 (pigz)	12.2	401.9	406.0	27.2	532.5	380.1	29.0	64.5	34.8
tar.bz2 -9	9.5	385.4	404.9	27.3	535.5	384.5	18.9	52.0	27.7
tar.zst -3 (dagens)	11.5	362.9	406.3	27.0	533.9	381.8	22.4	63.1	33.8
tar.zst -19 --long	8.5	252.7	405.1	14.2	**532.0**	261.4	13.1	50.2	25.2
tar.zst -22 --ultra	8.5	245.2	405.1	14.2	532.0	242.8	12.9	49.8	24.1
tar.xz -9e	8.2	266.4	405.4	14.2	532.9	251.7	11.8	46.2	23.7
nvz snabb	11.3	279.3	408.5	14.4	536.8	329.0	22.4	63.2	33.8
nvz utan knep	8.2	256.5	406.4	14.2	532.0	242.5	11.9	46.1	23.6
nvz normal	8.2	219.8	**345.1**	**14.2**	532.0	242.5	11.9	46.1	23.6
nvz liten	**7.2**	**207.2**	**345.1**	**14.2**	532.0	**237.0**	**10.0**	**43.5**	**20.4**

MiB per corpus; column minimum in bold. Raw sizes: kod 55, png 431, foto 409, gif 27, film 537, arkiv 392, logg 238, silesia 202, enwik8 95 MiB.

Pack time

compressor	kod	png	foto	gif	film	arkiv	logg	silesia	enwik8
tar (okomprimerad)	**0**	3	2	0	1	1	0	**0**	**0**
tar.gz -6	2	20	19	1	24	18	4	12	7
tar.gz -9 (pigz)	0	2	1	0	2	1	1	2	1
tar.bz2 -9	7	74	57	4	78	56	32	24	11
tar.zst -3 (dagens)	0	**1**	**0**	**0**	**0**	**0**	**0**	0	0
tar.zst -19 --long	35	40	53	4	57	47	113	63	61
tar.zst -22 --ultra	56	156	225	5	266	154	986	179	118
tar.xz -9e	47	128	157	8	154	144	897	158	121
nvz snabb	1	2	2	0	3	2	2	1	1
nvz utan knep	40	42	4	4	57	146	854	92	122
nvz normal	40	127	48	6	56	153	851	92	122
nvz liten	68	280	55	6	57	323	191	100	129

s per corpus; column minimum in bold. Raw sizes: kod 55, png 431, foto 409, gif 27, film 537, arkiv 392, logg 238, silesia 202, enwik8 95 MiB.

Uppack time

compressor	kod	png	foto	gif	film	arkiv	logg	silesia	enwik8
tar (okomprimerad)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
tar.gz -6	0	3	2	0	2	2	1	1	1
tar.gz -9 (pigz)	0	3	2	0	2	2	1	1	1

compressor	kod	png	foto	gif	film	arkiv	logg	silesia	enwik8
tar.bz2 -9	2	44	41	3	54	39	6	10	6
tar.zst -3 (dagens)	0	0	**0**	**0**	0	1	0	0	0
tar.zst -19 --long	0	1	0	0	0	0	0	0	0
tar.zst -22 --ultra	0	1	0	0	1	1	0	0	0
tar.xz -9e	1	5	4	0	5	7	1	3	1
nvz snabb	0	1	1	0	1	1	1	1	1
nvz utan knep	1	1	1	0	1	1	2	2	2
nvz normal	1	8	18	0	1	1	2	2	2
nvz liten	67	11	17	0	1	8	198	99	132

s per corpus; column minimum in bold. Raw sizes: kod 55, png 431, foto 409, gif 27, film 537, arkiv 392, logg 238, silesia 202, enwik8 95 MiB.

3.3 Per corpus

Complete series per corpus. *Mot dagens* ("vs. current") is the size change relative to `tar.zst -3`; *exakt* is the bit-exactness outcome.

Images (GIF) — 27.4 MiB i 105 filer

compressor	size MiB	kvot	vs. current	pack time	uppack time	exact
tar (okomprimerad)	27.62	0.99×	+2.1 %	0 s	0 s	yes
tar.gz -6	27.18	1.01×	+0.5 %	1 s	0 s	yes
tar.gz -9 (pigz)	27.19	1.01×	+0.6 %	0 s	0 s	yes
tar.bz2 -9	27.28	1.00×	+0.9 %	4 s	3 s	yes
tar.zst -3 (dagens)	27.04	1.01×	+0.0 %	**0 s**	0 s	yes
tar.zst -19 --long	14.18	1.93×	−47.6 %	4 s	0 s	yes
tar.zst -22 --ultra	14.18	1.93×	−47.6 %	5 s	0 s	yes
tar.xz -9e	14.19	1.93×	−47.5 %	8 s	0 s	yes
nvz snabb	14.44	1.90×	−46.6 %	0 s	0 s	yes
nvz utan knep	14.18	1.93×	−47.6 %	4 s	0 s	yes
nvz normal	**14.18**	1.93×	−47.6 %	6 s	0 s	yes
nvz liten	**14.18**	1.93×	−47.6 %	6 s	0 s	yes

Video (h264 and VP9) — 536.7 MiB i 67 filer

compressor	size MiB	kvot	vs. current	pack time	uppack time	exact
tar (okomprimerad)	536.82	1.00×	+0.6 %	1 s	0 s	yes
tar.gz -6	532.39	1.01×	−0.3 %	24 s	2 s	yes
tar.gz -9 (pigz)	532.54	1.01×	−0.2 %	2 s	2 s	yes
tar.bz2 -9	535.53	1.00×	+0.3 %	78 s	54 s	yes
tar.zst -3 (dagens)	533.86	1.01×	+0.0 %	**0 s**	0 s	yes

compressor	size MiB	kvot	vs. current	pack time	uppack time	exact
tar.zst -19 --long	**532.02**	1.01×	−0.3 %	57 s	0 s	yes
tar.zst -22 --ultra	532.03	1.01×	−0.3 %	266 s	1 s	yes
tar.xz -9e	532.86	1.01×	−0.2 %	154 s	5 s	yes
nvz snabb	536.75	1.00×	+0.5 %	3 s	1 s	yes
nvz utan knep	532.04	1.01×	−0.3 %	57 s	1 s	yes
nvz normal	532.04	1.01×	−0.3 %	56 s	1 s	yes
nvz liten	532.04	1.01×	−0.3 %	57 s	1 s	yes

Archives (zip and tar.gz) — 391.8 MiB i 133 filer

compressor	size MiB	kvot	vs. current	pack time	uppack time	exact
tar (okomprimerad)	391.93	1.00×	+2.7 %	1 s	0 s	yes
tar.gz -6	380.05	1.03×	−0.5 %	18 s	2 s	yes
tar.gz -9 (pigz)	380.14	1.03×	−0.4 %	1 s	2 s	yes
tar.bz2 -9	384.49	1.02×	+0.7 %	56 s	39 s	yes
tar.zst -3 (dagens)	381.78	1.03×	+0.0 %	**0 s**	1 s	yes
tar.zst -19 --long	261.37	1.50×	−31.5 %	47 s	0 s	yes
tar.zst -22 --ultra	242.77	1.61×	−36.4 %	154 s	1 s	yes
tar.xz -9e	251.75	1.56×	−34.1 %	144 s	7 s	yes
nvz snabb	329.03	1.19×	−13.8 %	2 s	1 s	yes
nvz utan knep	242.51	1.62×	−36.5 %	146 s	1 s	yes
nvz normal	242.51	1.62×	−36.5 %	153 s	1 s	yes
nvz liten	**237.00**	1.65×	−37.9 %	323 s	8 s	yes

Log files — 238.3 MiB i 76 filer

compressor	size MiB	kvot	vs. current	pack time	uppack time	exact
tar (okomprimerad)	238.39	1.00×	+965.0 %	0 s	0 s	yes
tar.gz -6	30.46	7.82×	+36.1 %	4 s	1 s	yes
tar.gz -9 (pigz)	29.03	8.21×	+29.7 %	1 s	1 s	yes
tar.bz2 -9	18.92	12.60×	−15.5 %	32 s	6 s	yes
tar.zst -3 (dagens)	22.38	10.65×	+0.0 %	**0 s**	0 s	yes
tar.zst -19 --long	13.09	18.21×	−41.5 %	113 s	0 s	yes
tar.zst -22 --ultra	12.92	18.45×	−42.3 %	986 s	0 s	yes
tar.xz -9e	11.84	20.12×	−47.1 %	897 s	1 s	yes
nvz snabb	22.41	10.63×	+0.1 %	2 s	1 s	yes
nvz utan knep	11.87	20.08×	−47.0 %	854 s	2 s	yes
nvz normal	11.87	20.08×	−47.0 %	851 s	2 s	yes
nvz liten	**10.03**	23.77×	−55.2 %	191 s	198 s	yes

3.4 Standard corpora, codec level

This comparison isolates the nvcm compressor — one file at a time, one thread — since published figures for these corpora are defined that way. nvz as an archiver appears in 3.1–3.3.

Silesia — 211,938,580 byte

codec	size bytes	kvot	bits/byte	pack MiB/s	unpack MiB/s	exact
gzip -9	67,631,990	3.134×	2.553	6.8	157.9	yes
bzip2 -9	54,506,769	3.888×	2.058	8.1	19.2	yes
zstd -19	52,895,350	4.007×	1.997	1.6	374.3	yes
zstd -22 --ultra	52,369,178	4.047×	1.977	1.2	360.9	yes
brotli -q 11	49,564,563	4.276×	1.871	0.3	175.8	yes
xz -9e	48,456,004	4.374×	1.829	1.3	66.5	yes
nvcm	44,439,240	4.769×	1.677	0.9	0.8	yes

enwik8 — 100,000,000 byte

codec	size bytes	kvot	bits/byte	pack MiB/s	unpack MiB/s	exact
gzip -9	36,445,248	2.744×	2.916	10.2	138.2	yes
bzip2 -9	29,008,758	3.447×	2.321	8.7	16.5	yes
zstd -19	26,944,227	3.711×	2.155	1.1	340.6	yes
brotli -q 11	25,742,001	3.885×	2.059	0.3	207.3	yes
zstd -22 --ultra	25,272,471	3.957×	2.022	0.8	381.5	yes
xz -9e	24,831,648	4.027×	1.986	0.8	69.6	yes
nvcm	21,417,184	4.669×	1.713	0.8	0.7	yes

Per file on Silesia, whose heterogeneity is the point of the corpus:

file	contents	raw bytes	zstd -19	brotli -q 11	xz -9e	nvcm	nvcm vs. best
`dickens`	English literature, text	10,193,352	2,849,945	2,827,777	2,831,212	2,234,894	−20.2 %
`mozilla`	executables (tar)	51,215,653	15,071,672	13,872,265	13,376,240	14,366,242	+7.4 %
`mr`	medical 3D image	9,970,424	3,106,737	2,823,136	2,751,892	2,205,358	−9.7 %
`nci`	chemical structures, text	33,553,317	1,664,988	1,519,768	1,449,272	1,505,038	+3.8 %
`ooffice`	executable DLL	6,151,755	2,595,377	2,478,855	2,427,224	2,379,789	−2.0 %
`osdb`	database dump, binary	10,084,646	3,100,177	2,816,278	2,844,556	2,395,972	−14.5 %
`reymont`	Polish literature, PDF source	6,627,493	1,348,462	1,332,158	1,315,592	1,053,488	−15.5 %
`samba`	source code (tar)	21,606,321	3,894,675	3,766,340	3,739,524	3,377,043	−9.7 %
`sao`	star catalogue, binary	7,250,892	5,000,576	4,586,092	4,425,664	4,509,261	+1.9 %
`webster`	dictionary, text	41,458,740	8,679,705	8,428,575	8,368,672	6,317,041	−24.5 %
`x-ray`	X-ray, 16-bit	8,475,800	5,129,827	4,682,753	4,491,264	3,742,075	−7.6 %
`xml`	XML	5,345,363	453,209	430,566	434,892	353,039	−18.0 %

As a reference against published results (LTCB; other machines and program versions; not our measurements):

program	enwik8 (bytes)	bits/byte	method
gzip 1.3.5	36,445,248	2.92	dictionary
bzip2 1.0.2	29,008,736	2.32	block sorting
zstd 0.6.0	25,405,601	2.03	dictionary
xz 5.2.1	24,703,772	1.98	dictionary (LZMA)
nanozip 0.09a	18,594,163	1.49	context mixing
zpaq 6.42	17,855,729	1.43	context mixing
paq8px_v206fix1	15,849,084	1.27	context mixing
cmix v21	14,623,723	1.17	context mixing

Our own measurement: **nvcm compresses enwik8 to 21,417,184 bytes, 1.713 bits/byte** — 13.8 % smaller than `xz -9e` on the same machine (24,831,648 bytes, 1.986 bpb).

Med 1.71 bits/byte hamnar nvcm mellan ordbokspackarna och de tunga kontextblandarna i tabellen ovan: den tar hem det en ordbokspackare inte kan se, men den har åtta modeller där cmix har hundratals, och den kör i ett par MiB/s där cmix kör i kilobyte per sekund.

3.5 Throughput

Seconds are not comparable across corpora of different size; the table therefore reports MiB/s of raw data through the wall clock.

corpus	nvz normal pack	nvz normal unpack	zstd -3 pack	zstd -19 pack	xz -9e pack
Source code	1	97	326	2	1
Images (PNG)	3	51	743	11	3
Photos (JPEG)	9	23	869	8	3
Images (GIF)	5	304	304	6	3
Video (h264 and VP9)	10	378	1,220	9	3
Archives (zip and tar.gz)	3	353	1,059	8	3
Loggfiles	0	125	1,254	2	0
Silesia (archive level)	2	82	612	3	1
enwik8 (archive level)	1	49	502	2	1

MiB per second, raw data through the wall clock.

3.6 LZMA2 at full strength

`xz -9e` in the tables above is LZMA2 with a 64 MiB dictionary. The format's full form is a different opponent: 7-Zip at `-mx=9` with a **512 MiB dictionary** and solid archive — the dictionary then covers each corpus in full except video, the same playing field as nvz's large blocks. The comparison ran as complete cycles (`experiment/lzma2-extrem.sh`), no filters, LZMA2 alone against nvz in its default mode:

corpus	raw MiB	LZMA2-extreme MiB	nvz MiB	nvz vs. LZMA2-extreme	pack time LZMA2/nvz	exact
Source code	55	8.22	**7.28**	−11.4 %	19 / 70 s	yes
Loggfiles	238	14.16	**10.03**	−29.2 %	70 / 196 s	yes
Arkiv (zip/tar.gz)	392	237.12	242.53	+2.3 %	43 / 195 s	yes

corpus	raw MiB	LZMA2-extreme MiB	nvz MiB	nvz vs. LZMA2-extreme	pack time LZMA2/nvz	exact
Images (PNG)	431	242.99	**208.81**	−14.1 %	40 / 268 s	yes
Silesia	202	46.43	**43.54**	−6.2 %	78 / 102 s	yes
enwik8	95	23.65	**20.43**	−13.6 %	79 / 132 s	yes

Two observations. nvz's lead on text persists against even the strongest LZMA2 form — context mixing collects what a larger dictionary cannot. On the archive corpus, however, LZMA2 at full strength beats zstd −22 and thus became the new best classical reference there: **nvz trailed by +2.3 %**. 7-Zip also packs consistently faster (multithreaded LZMA2).

Remedied 2026-11-18. The gap had a mechanism, not merely a number: nvz's LZMA codec ran `preset 9e`'s fixed 64 MiB dictionary, while 7-Zip at maximum had 512 MiB and therefore covered the whole block. nvz's dictionary is now scaled to the block (see 4.2 and the comment in `src/cpp/kodek.cpp`), and LZMA was given a place as a candidate in the opaque escalation. Re-measured, nvz is 0.05 % **below** LZMA2 at maximum on the archive corpus — 248,512,199 against 248,636,548 bytes. The archive row in the table above is therefore out of date; it stands as measured on 2026-11-17, and the re-measurement is in 4.2.

3.7 The Linux kernel: a corpus that is not ours

Every corpus above was chosen by us, and the objection that a result may be tuned to its test data cannot be answered with more of the same data. `experiment/karntest.sh` therefore runs the whole round trip on a third-party tree: Linux kernel 6.12, downloaded from `cdn.kernel.org`, with a checksum we did not set ourselves. No threshold and no parameter in nvz was chosen with it in view.

packer	bytes	vs. kernel.org's own	pack time
kernel.org `tar.xz` (as shipped)	147,906,904	±0	—
`tar.zst -19 --long`	147,274,821	-0.4 %	117 s
`7z` LZMA2 `-mx=9`, 512 MiB dictionary, solid	141,350,334	-4.4 %	264 s
`tar.xz -9e` (local, 64 MiB dictionary)	140,072,812	-5.3 %	304 s
`nvz --läge normal`	141,787,458	-4.1 %	201 s
** `nvz --läge liten` **	**123,217,795**	**-16.7 %**	443 s

1,476,500,559 bytes in 86,618 files. Unpacking `liten`: 439 s.

Against the best classical tool on the same machine nvz is 12.0 % smaller, and against LZMA2 in its maximum form (3.6) 12.8 % smaller. `normal` trails `xz -9e` by 1.2 % but packs 1.5 times faster. That the outcome (-12.0 %) is of the same order as our own code corpus (-11.6 %, 3.1) is the real result of this test: the margin carries over to data we did not choose.

The correctness half was run as a manifest before and after, checked by `md5sum -c` rather than by nvz's own code: **86,618 of 86,618 files OK, 0 FAILED**, with the manifests identical line by line.

3.8 zlib-ng/corpora: eight corpora chosen by someone else

The suggestion came from outside, and the value lies in the selection: `zlib-ng/corpora` collects the literature's old standard corpora — Calgary and Canterbury — alongside `artificial` (pathological cases), `neuro` (measurement data), `snappy`, `large` and Silesia. Calgary and Canterbury consist of **small** files, which is where nvz is weakest (section 8, item 3), so the selection is not a favourable one. Every row was run as a full round trip with bit-exact comparison.

corpus	raw	gzip -9	bzip2 -9	zstd -19	xz -9e	nvz `normal`	nvz `liten`	nvz vs. best classical
artificial	300,001	77,226	76,304	75,598	77,128	75,521	**75,521**	**-0.1 %**
calgary	3,251,493	1,061,241	893,979	896,593	855,732	851,964	**726,530**	**-15.1 %**
canterbury	2,788,958	730,537	565,459	509,973	479,812	481,525	**404,235**	**-15.8 %**
large	11,094,363	3,193,076	2,598,816	2,596,688	2,557,956	2,542,474	**2,222,537**	**-13.1 %**
miscellaneous	1,000,000	470,636	431,450	428,047	436,608	426,341	**417,715**	**-2.4 %**
neuro	57,213,632	26,878,246	24,253,357	25,070,122	22,026,428	22,468,734	22,243,418	**+1.0 %**
silesia	210,806,959	67,458,471	54,586,956	52,501,333	48,355,076	48,301,220	**45,655,803**	**-5.6 %**
snappy	1,742,488	547,936	459,545	419,541	404,200	385,195	**372,174**	**-7.9 %**

Bytes per corpus, the whole subdirectory as a tree. The round trip was bit-exact in all 24 runs (three modes × eight corpora).

Seven of eight corpora go to nvz, with -15.8 % on Canterbury as the best outcome. The eighth is `neuro`, where nvz trails `xz -9e` by **1.0 %**: the material is binary measurement data without the between-file similarity that nvz's transforms and deduplication live on, leaving only the codec — and there LZMA with a 64 MiB dictionary is stronger than nvcm on that particular data. It is the same pattern as the archive corpus in 4.2 and should be read as the boundary of H1: nvz improves data with exploitable structure, and on data without it the result is a bare percent worse, not better.

3.9 Dominance: smaller AND faster, corpus by corpus

The tables above compare size and time separately. The harder question is whether nvz can beat the **same** opponent on both axes at once — a requirement raised on 2026-11-18: *better than the established tools, everywhere, faster and at least as good.* `experiment/dominans.py` tests it: the opponent per corpus is whichever established packer produced the smallest archive, with its own measured pack time on the same machine, and of nvz's modes the smallest of those that dominate is counted.

```
korpus     bästa etablerade                  byte    sek | nvz              byte    sek | utfall
kod        tar.xz -9e                   8 629 516     47 | normal       8 580 323     40 | DOMINERAR
(-0.57 % storlek, -15 % tid)
png        tar.zst -22 --ultra        257 132 369    156 | normal     230 452 176    127 | DOMINERAR
(-10.38 % storlek, -18 % tid)
foto       tar.bz2 -9                 424 611 032     57 | normal     361 834 581     48 | DOMINERAR
(-14.78 % storlek, -16 % tid)
gif        tar.zst -22 --ultra         14 867 185      5 | liten       14 864 575      6 | mindre, men
långsammare (-0.02 % storlek, +20 % tid)
film       tar.zst -19 --long         557 861 782     57 | normal     557 880 433     56 | snabbare, men
större (+0.00 % storlek, -0 % tid)
arkiv      tar.zst -22 --ultra        254 567 277    154 | normal     254 287 214    153 | DOMINERAR
(-0.11 % storlek, -1 % tid)
logg       tar.xz -9e                  12 419 456    897 | liten       10 512 452    191 | DOMINERAR
(-15.35 % storlek, -79 % tid)
silesia    tar.xz -9e                  48 408 228    158 | liten       45 657 473    100 | DOMINERAR
(-5.68 % storlek, -37 % tid)
enwik8     tar.xz -9e                  24 831 492    121 | liten       21 417 335    129 | mindre, men
långsammare (-13.75 % storlek, +7 % tid)

dominerar på 6 av 9 korpusar
```

The pattern is not random. nvz dominates where there is structure to exploit — source code, logs, photographs, images, mixed trees — and falls short on TIME, never on size, where there is nothing to exploit: film and gif are already entropy-coded, the archive corpus needs an entire extra LZMA pass for its last percent, and enwik8 is a single text file where neither deduplication, sorting nor transforms have anything to contribute and nvz's five sweeps over the data (README, "The floor is not the codec") become pure cost.

The boundary therefore does not run between data types but between *exploitable structure and none*: the same conclusion as H1, but measured on two axes instead of one.

---

4. Analysis

4.1 Log data: compressor choice dominates

The log corpus exhibits the evaluation's highest ratio, 23.8×, and the source can be isolated through the mode comparison: `normal` and `liten` differ only in whether nvcm participates in the codec probe.

mode	size	pack time	active difference
`normal`	11.97 MiB	864 s	codec probe zstd/xz
`liten`	**10.03 MiB**	**197 s**	nvcm added

nvcm won the probe for every text block, which the archive register confirms (2 blocks, both nvcm; 192.03 → 9.00 MiB and 46.29 → 1.03 MiB). That the smaller archive also packs faster is a parallelism effect: nvz's blocks compress independently, whereas `xz -9e -T0` on this input effectively runs single-threaded due to its per-thread memory budget. The blocks' differing ratios (21.4× vs. 45.1×) illustrate adaptation: a model that has seen 192 MiB of syslog predicts the next block substantially better.

4.2 Already-compressed archives: the limit of sampling

On the archive corpus, nvz is 20.0 % larger than `zstd −22 --ultra --long=31` (291.3 vs. 242.8 MiB). Two explanatory hypotheses were tested experimentally (`experiment/arkiv-fonster.sh`): (a) block partitioning prevents cross-block matching; (b) the codec probe selects the wrong codec.

configuration	size
zstd −19, default window (8 MiB)	365.3 MiB
zstd −19 `--long=27`, whole stream	261.4 MiB
zstd −22 `--ultra --long=31`, whole stream	242.8 MiB
nvz, probe selects	291.3 MiB
nvz, `--kodek zstd19`	260.2 MiB
nvz, `--kodek zstd22`	**244.6 MiB**

Hypothesis (a) is falsified: with a manually chosen codec, partitioning costs 1.2 MiB (0.7 %). Hypothesis (b) is confirmed: the probe compresses 4 MiB sampled across the block, while the corpus's redundancy lies in nearly identical files hundreds of megabytes apart — a property a sparse sample cannot observe. The probe therefore classifies the block as poorly compressible and selects a cheap codec. Deduplication captures the exact copies (62.8 MiB, 16.0 %); the remainder consists of *near*-copies (distinct gzip streams of similar raw data), visible only to long-range matching over the whole stream.

Remedy and re-measurement. The diagnosis was implemented as two rules: (i) opaque blocks are not split at 192 MiB — on opaque data, block partitioning is a compression bound rather than a memory optimization, and zstd parallelizes within the block; (ii) for opaque blocks the sample classifies as compressible, the sampled choice is replaced by a two-stage rule, since a sample cannot distinguish video from archives-of-archives — both are locally dense, and the difference only shows over the whole block. The block is first compressed with zstd −19 (2^{27} window); a gain above 10 % indicates inter-file redundancy, and the block is recompressed with zstd −22 (2^{31} window), which collects the remainder. Video stops at around one percent and never pays for the expensive path. Re-measured:

configuration	size
zstd –22 `--ultra --long=31`, whole stream	242.8 MiB
nvz after remedy (1 block, escalated to zstd22)	**242.5 MiB**

The video corpus exercises the rule's other branch: escalation does not trigger (zstd −19 gains 0.9 %), and pack time stays at 60 s against the 255 s an unconditional zstd −22 would have cost.

nvz thus moves from +20.0 % to −0.1 % against the best classical tool on this corpus; the margin is deduplication's 62.8 MiB, which the classical pipeline must rediscover as matches rather than never storing.

Second re-measurement, 2026-11-18. The two-step rule had two weaknesses that surfaced when the question was raised whether nvz can beat the best established tool *everywhere*:

The cheap step was not cheap. The rule used `zstd -19` to answer the question "does this block have tens of percent to give?". But that question needs no strong compressor, only an honest run over the whole block. Measured on the gif corpus:

codec over the whole block	size	time
`zstd -3`	14,983,720	**68 ms**
`zstd -19`	14,868,267	4,388 ms
`zstd -22`	14,867,123	3,856 ms

`zstd -3` sees 47.8 of the 48 percentage points available, in one sixtieth of the time. The first step is therefore `zstd -3`. The branch for film-like data keeps `zstd -19` as its final step, since it is both cheaper and better than `zstd -22` on such data (film: 557,863,118 bytes in 56 s versus 557,873,791 in 264 s).

The expensive branch lacked its strongest candidate. Where the cheap step has shown that the block holds between-file similarity, mode `liten` now also tries **LZMA with a dictionary covering the whole block** — the only candidate whose window can reach across an opaque block of several hundred megabytes, and precisely the form 7-Zip beat nvz with in 3.6. With it:

configuration	size	vs. nvz
`zstd -22 --ultra --long=31`, whole stream	254,310,751	+2.3 %
7-Zip LZMA2 `-mx=9`, 512 MiB dictionary, solid	248,636,548	+0.04 %
nvz `liten` after the remedy	**248,512,199**	—

nvz is thus **0.04 % smaller than LZMA2 in its maximum form** on the corpus that was the evaluation's only counter-instance to H1. The price is time: 188 s against 7-Zip's 43 s, since the step is an entire additional compression of the block. That trade-off is framed by the modes — `normal` does not make it, `liten` does.

4.3 Video: the null result and its price

All twelve configurations land within 1 % of raw size — expected, since transform-coded video has no entropy layer accessible to generic compression. The relevant comparison is thus the cost of establishing this: nvz `normal` reaches the same size as `zstd −22` in 16 % of the time (61 vs. 266 s), because the codec probe classifies the blocks as raw-storage cases from samples instead of compressing them in full.

4.4 GIF: ratio without compression contribution

The GIF corpus's 1.93× ratio is fully explained by duplicates (47.3 % of the corpus) and by the corpus (27.4 MiB) fitting inside every modern compressor's window: nvz, zstd −19/−22 and xz reach the same size, and `nvz utan knep` is identical to `nvz normal`. nvz's transforms contribute nothing on LZW-coded content — the format is not

opened. A GIF counterpart to nvjpg (LZW → arithmetic coding of the frames) is thus an identified but unmeasured opportunity; the corpus to measure it now exists.

4.5 The dictionary, the memory and the parallelism

Letting LZMA's dictionary cover the block (4.2) gained 2.17 % on archive data. But the change had a second-order effect visible only on a machine short of memory, and it was worse than the gain.

The encoder needs roughly ten times the dictionary. The PNG corpus's blocks are 192 MiB and therefore received a 256 MiB dictionary and a memory requirement of 2.7 GiB each. Nine such blocks do not fit in free memory, so the gate admitted **one at a time**: pack time went from 268 to 841 seconds for an archive 0.06 % smaller. A dictionary gain that removes the parallelism is not a gain.

Three corrections, ordered by what they gave:

change	PNG corpus pack time
dictionary covering the block, with no regard for memory	841 s
dictionary shrunk until it fits the budget divided by the number of concurrent encoders	191 s
codec probe run per class instead of per block	178 s
dictionary covers the block but never exceeds it, and the budget is all free memory	**129 s**

The last row is the most important and the most obvious in hindsight: a 32 MiB block received a 64 MiB dictionary because the floor had been set to `preset 9e`'s value. A dictionary larger than the block cannot be used by construction — but it costs memory, and memory is what decides how many blocks are compressed at once. The floor is therefore gone: the dictionary is exactly what the block needs, rounded up to a power of two.

In addition, the class `png-ra` — unfolded PNG filter rows — is split at 32 MiB in every mode but `liten`. The class is special in three ways: it is the only one that is inflated (431 MiB of files become 2 GiB of blocks), its similarity lies between neighbouring images rather than hundreds of megabytes away, and it is therefore the class where splitting costs the least ratio per second saved:

block cap for png-ra	pack time	archive
192 MiB (`liten`)	280 s	217,308,163
32 MiB (`normal`)	**127 s**	230,452,176

The best established tool on this corpus is `zstd -22` at 257,132,369 bytes in 156 s. `normal` is therefore 10.4 % smaller **and** 18 % faster, while `liten` still takes the last 5.7 % for those who want them. That is the trade-off the modes exist for, now measured at both ends.

The price of this whole section is on the record too: nvz's dictionary — and with it both ratio and time — now depends on how much memory the machine has free. Measurements must therefore be made on an otherwise unloaded machine, or one measures the gate rather than the compression.

5. Correctness

5.1 Restoration

Recursive diff answers a weaker question than H2 requires: it compares contents of files that exist on both sides, but is blind to permissions, timestamps, empty directories, and entries silently dropped. `prov/aterskapning.py` therefore computes a complete fingerprint — SHA-256, mode, mtime in nanoseconds, link target, entry type —

before packing and after unpacking, requires identity, and verifies that the archive register's entry count matches the tree's.

corpus	files	register entries	SHA-256 equal	mode and times equal	hard links left	deviations
gif	105	yes	ja	yes	no	none
arkiv	133	yes	ja	yes	no	none
logg	76	yes	ja	yes	no	none
film	67	yes	ja	yes	no	none
png	4,618	yes	ja	yes	no	none
foto	1,143	yes	ja	yes	no	none
kod	5,974	yes	ja	yes	no	none

Damage tests on the archive from corpus **gif**:

damage	outcome	passed
magin bytt	rejected	yes
byte i blockkropp	rejected	yes
sista kilobyten bort	rejected	yes
registerpekaren noll	rejected	yes
en byte i registret	rejected	yes

In addition, `prov/prova-allt.sh` exercises edge cases — empty files, duplicates, relative, broken, and directory-targeting symlinks, filenames with newlines and non-ASCII, permissions 400–755, timestamps in 1985 and 2031 — in every mode, requiring full metadata identity. The suite comprises 44 tests and passes without exception.

The edge-case suite, however, only exercises the cases someone thought of. `prov/slumpprov.py` exercises combinations nobody thought of: randomised file sizes, file types, duplicate patterns, permissions, times, names and symlinks, with real PNGs and JPEGs from the corpus mixed in, each round being a full round trip in every mode plus the archive's own check. **400 rounds (seeds 5,000–5,399) came back bit-exact, zero deviations.** A failing round prints its seed so the fault can be reproduced exactly — and that mechanism is what caught the build fault in section 6, item 4, on its first round.

5.2 Damage tolerance

An archiver that returns incorrect data without a failure indication is worse than one that refuses: the former is discovered only at use. Five damage classes were injected into a healthy archive; the requirement is rejection or provably intact output, never silent corruption.

damage class	outcome
replaced magic	rejected
bit flip in block body	rejected
truncation (final 1,024 bytes)	rejected
zeroed register pointer	rejected
bit flip in register	rejected

5.3 Version compatibility

The C++ implementation (section 6) reads the Python implementation's archives (JSON register, 8-byte hash prefix) bit-exactly, including PNG- and JPEG-transformed objects; this is part of the test suite. The reverse is unsupported: the v2 register is binary, and the Python implementation is retired.

Register version 3 (2026-11-18). The register now carries three changes, all made so that the archive would come out smaller than the best established tool's on file-rich trees:

change	why	gain on the code corpus
object hash 16 → 4 bytes	the hash is used by `-t` to detect mis-assembled content, and 32 bits suffice for that; bit errors are already caught by the blocks' CRC64/xxhash. Hashes are random and therefore incompressible	70,776 bytes
time as epoch seconds instead of an ISO string	5 bytes instead of 20	15 bytes
the register is compressed with whichever of xz and zstd is smaller	the register is text and numbers; xz handles it better, but its container is ~60 bytes against zstd's ~13 and therefore loses on small archives	12,005 bytes

The register went from 239,049 to 153,596 bytes — from 2.76 % of the archive to 1.79 % — and the whole archive from 8,675,609 to 8,578,823 bytes. **Reading is backward compatible:** v1 (JSON), v2 (16-byte hash, zstd) and v3 are all read by the same binary, and which compressor the register used is decided by the stream's own magic rather than by the version field, which sits inside the compressed register. The reverse does not hold: an nvz built before 2026-11-18 cannot read a v3 archive. The test suite covers all three versions.

5.4 The evaluation's own error source: a living corpus

During the re-measurement on 2026-11-18, two rows reported "SKILJER" on the `kod` corpus — the recursive comparison found three files that did not come back identical. The fault was not in nvz.

`korpus/bygg-korpus.py` **hard-links** rather than copies, to save disk. The file in `korpus/kod` is therefore the same inode as the file in the live working tree, and that holds for nearly everything:

corpus	files with more than one link
kod	5,971 of 5,974
png	4,618 of 4,618
foto	1,143 of 1,143
gif	105 of 105
film	67 of 67
arkiv	131 of 133
logg	0 of 76

The three files sat in `korpus/kod/ullabella/`, and their mtimes fell between the compression and the comparison: someone saved in another window while the measurement was running. Fifteen files in the corpus had changed during the preceding two hours. The archive was therefore flawless; it was the corpus that had moved.

The log corpus has always been copied for exactly this reason — it says so in `bygg-korpus.py`'s own header: *"unpacking makes every comparison meaningless"*. The conclusion applies to every corpus, not only to logs. Two remedies:

1. `matning/matning.py` now takes a cheap fingerprint of the corpus (path, size and mtime per file, no content read) before and after each measurement, and writes the field `korpus_stabil` into the result. A measurement

during which the corpus moved is invalid and must be repeated — not interpreted.

2. `korpus/bygg-korpus.py --kopiera` builds the corpus from real copies instead of hard links. It costs roughly 2.3 GiB of disk and makes the corpus hold still.

The two rows were re-measured on a stable corpus and are bit-exact. But the consequence is broader than two rows: **every figure in this report measured before that day rests on a corpus that could move.** That the fault has surfaced only once, in a comparison that caught it, is an argument that the test suite is correctly built — not that the corpus was.

---

6. Implementation

During the evaluation, nvz was rewritten from Python into a standalone C++17 binary (367 kB, 318 kB stripped): nvcm and nvjpg are linked in and executed as worker processes via `/proc/self/exe`, zstd/xz/zlib are called as in-process libraries, and `liten` is the default mode. Compression results are equivalent within sampling noise (the codecs are the same libraries); the difference is the time profile wherever Python participated per file:

data point	Python	C++
code, mode `snabb` (5,974 files)	2.7 s	0.8 s
gif, mode `snabb`	0.6 s	0.1 s
516 JPEG transforms	≈8 s	2.8 s
photo, unpack (1,143 transformed)	7.8 s	17.4 s

The remaining regression on photo unpacking stems from worker-process startup cost per image and is the single point where the Python implementation's process pool was faster.

The port exposed three defects of the kind that only manifest under concurrency or format migration, documented here for their generality:

1. **Allocation between `fork` and `exec` in a threaded process.** The child inherits lock state from threads that do not survive the fork; an allocation in the child can then block permanently. Symptom: workers at 0 % CPU for 84 minutes. Remedy: the argument vector is built before `fork`; between `fork` and `exec` only async-signal-safe calls occur.
2. **File-descriptor inheritance between parallel workers.** A child forked while another thread's pipes are open inherits them too; the peer's read end then never sees EOF. Remedy: `pipe2(O_CLOEXEC)`.
3. **Hash length in the v1 register.** The Python register stores the first 16 hex characters of the object hash — 8 bytes. A reader comparing 16 bytes rejects healthy archives.
4. **Incomplete header dependencies in the build.** The object-file rule read `%.o: %.cpp src/cpp/nvz.hpp` and therefore tracked a single header. A field added to `struct Val` (arkiv.hpp) rebuilt `huvud.o` and `arkiv.o` but not `yta.o`, and the binary was linked from object files that disagreed about the struct's layout. The result was a binary where `nvz packa` worked flawlessly while `nvz -t` crashed with SIGSEGV — no compiler warning, no link error. The fault was found by `prov/slumpprov.py` (section 5) on its first round and is the clearest illustration of why the randomized test exists: it exercises the whole surface, not the path one happens to use by hand. Remedy: dependencies are generated by the compiler (`-MMD -MP`) and included in the Makefile.

The codec layer, 2026-11-18. Six changes, all with measurements in 4.2, 4.5 and 5.3: LZMA's dictionary is scaled to the block instead of being `preset 9e`'s fixed 64 MiB; the dictionary never exceeds the block and is further shrunk until it fits free memory divided by the number of concurrent encoders; `pb = 0` is tried as a variant for data without a word length; the codec probe is run per content class instead of per block; the cheap step of the opaque two-step rule is `zstd -3` instead of `zstd -19`; and in mode `liten` the choice is re-tested on the whole block when

the sample's margin was thin. The register is version 3 (5.3). All changes are backward compatible for READING — v1, v2 and v3 are read by the same binary — but a v3 archive cannot be read by an nvz built before that day.

1. **The boundary of the self-check, and three faults that lived outside it.** A code review by Ted Lyngmo on 2026-11-18 found three flaws that neither the test suite, the reconstruction tests nor the randomised test could see, and they shared one cause: those tests pack and unpack ON THE SAME MACHINE, in a successful flow, with readable files and free disk.

The PNG recipe is not portable. The recipe is four zlib parameters, and the assumption that they determine the output stream does not hold — the output depends on the implementation, and zlib-ng produces different bytes than zlib. An archive packed against one could not be unpacked against the other. Fixing it revealed something worse: `png_aterskapa` copied the skeleton's CRC straight into the output, so a stream of the right LENGTH but the wrong bytes would have produced a PNG with wrong image data and a correct-looking CRC — silent corruption. Each IDAT chunk is now verified against its CRC, the recipe carries a behavioural fingerprint of the deflate engine (fixed probe string, crc32 of the output), and on a miss the level, memory level and strategy are searched again; the window is not varied, as it is stated in the file's own zlib header. If the stream still cannot be reproduced, nvz refuses with both fingerprints in the message. What remains is that a PNG-heavy archive may need a machine with the same engine — that is the format's price and it belongs where it can be seen.

Write failures caused silent data loss. No write in the archive path was checked, and the gzip surface deletes the source after a successful pack. A full disk therefore produced a truncated archive, exit status 0 and a deleted original. Every `fwrite`, `fflush` and `fclose` is now checked, the file is `fsync`ed before closing, and an archive that did not finish is removed. SIGXFSZ is deliberately ignored: gzip dies at 153 and never cleans up, while nvz gets EFBIG and returns 1.

An unreadable file killed the process. `nvz a b c` with b unreadable escaped through `main` and died in `std::terminate` — SIGABRT, exit 134, with a's original already deleted and c never processed. Each operand is now caught on its own: warning, the rest are packed, exit status 1. That is gzip's contract, measured on the same case.

All three are regression tests in `prov/brister.sh`.

The file names in `src/cpp/` changed language on 2026-11-19. The text in this section was written against the Swedish names; the mapping is huvud→main, yta→cli, arkiv→archive, kodek→codec, omvandla→transform, trad→tree, sjalv→worker, verktyg→util. The code, the command line and the README are now English, with the Swedish command names kept as permanent aliases — so every command in this report still works unchanged.

7. Related work

One objection raised in the public discussion of this work belongs in the report rather than in a comment thread: *every individual method nvz uses is known and has been published for years.* That is correct. The table below credits each part, and the last column states where nvz sits relative to it.

nvz component	known prior art	nvz vs. the prior art
similarity sorting before compression	*Sort Before Compression* (<https://gwern.net/sort>); solid archives in RAR and 7-Zip; lrzip	same idea. nvz sorts on content class and extension, and measures the effect separately (7.1)
deduplication of identical files	borg, restic, ZFS; content-defined chunking since LBFS (2001)	weaker: nvz deduplicates whole files, not chunks
deflate stream back to raw data plus a recipe	precomp (Christian Schneider), preflate, reflate	narrower: nvz only handles streams zlib itself wrote
lossless JPEG recompression	packJPG (Matthias Stirner), Lepton (Dropbox, 2016), brunsli -> JPEG XL	**worse**: nvjpg achieves -15.1 %, published figures for the others are around -20 to -22 %

nvz component	known prior art	nvz vs. the prior art
context mixing, logistic mixer, APM/SSE	the PAQ family (Matt Mahoney), lpaq, zpaq, cmix	smaller model: paq8px beats nvcm on enwik8, 15.8 vs. 20.4 MB (3.4)
content-aware filter selection per data type	FreeArc (Bulat Ziganshin), zpaq method strings	same thought, different mechanism: nvz decides by probing the data rather than by a rule on file type
transform verified in both directions before use	precomp and Lepton do the same	same

The figures for packJPG, Lepton, brunsli and paq8px are **published values, not measured here**; the exception is paq8px on enwik8, which is part of 3.4.

What remains as this project's own is therefore not an algorithm but three things: that the parts sit in one tool with one archive format and one reconstruction guarantee; that the choice between codecs is settled by probing the data rather than by following the file extension; and the evaluation in this report, including the counter-instance it found and fixed (4.2). nvz does not beat information theory, and does nothing nobody has done: it packages known technique and reports what the packaging is worth.

7.1 How much comes from the sorting?

The question cannot be answered one archive at a time, because the tricks are measured together. `experiment/ablation.sh` therefore holds the **compressor constant** -- the same `xz -9e`, the same 64 MiB dictionary -- and enables one trick at a time via the flags `--utan-sortering`, `--utan-klassning`, `--utan-dedup` and `--utan-omvandling`. The `kod` corpus is 55 MiB and fits entirely inside the dictionary on every row, so window size is not a free variable: what differs between rows is the ordering and nothing else.

Ablation, corpus `kod` (55 MiB), codec `xz -9e` throughout. The whole corpus fits inside the dictionary (64 MiB), so the compressor sees every file from every other file on every row:

step	bytes	vs. tar	the step itself
tar in directory order (no nvz)	8,611,308	±0	—
nvz, all tricks off	8,673,944	+0.73 %	+0.73 %
+ one block per content class	8,673,942	+0.73 %	-0.00 %
+ sorting within the class	8,676,765	+0.76 %	+0.03 %
+ deduplication	8,675,644	+0.75 %	-0.01 %
+ transforms	8,675,644	+0.75 %	±0

Ablation, corpus `png` (431 MiB), codec `zstd -19 --long=27` throughout:

step	bytes	vs. tar	the step itself
tar in directory order (no nvz)	264,771,708	±0	—
nvz, all tricks off	268,949,955	+1.58 %	+1.58 %
+ one block per content class	268,949,973	+1.58 %	±0
+ sorting within the class	268,949,972	+1.58 %	-0.00 %
+ deduplication	258,596,586	-2.33 %	**-3.85 %**
+ transforms (png/jpeg)	227,591,646	**-14.04 %**	**-11.99 %**

The result is clear and partly uncomfortable. **Sorting measures ±0 on both corpora**, and so does class blocking. What does measure is the transforms (-12.0 % on png) and deduplication where duplicates exist (-3.9 % on png, where 28.7 % of the bytes are duplicates; ±0 on kod, where 0.7 % are). nvz's own archive layer also costs +0.7 to

+1.6 % on the rows where nothing is enabled — register, block headers, and the fact that blocks are compressed separately.

Zero is not the same as useless, however, and the cause of the zero is not the sorting but the corpora: `png` has a single file type, so sorting on (class, extension, path) yields the order directory order already had, and `kod` fits entirely inside the dictionary. A real project tree also has its similarity already grouped per directory — `.py` sits with `.py` in the same package — so there is no scatter to fix. With `zstd -3` (2 MiB window) on the same code corpus, sorting even measures slightly negative, +0.14 %: it breaks up a locality directory order already provided.

`experiment/sortering.sh` therefore constructs the case sorting exists for, out of exactly the same data: the same 5,974 files from `korpus/kod`, copied into 200 directories with randomised names (fixed seed), so that directory order loses all locality while the content is unchanged.

ordering	`zstd -3` (2 MiB window)	`xz -9e` (64 MiB dictionary)
tar in directory order	13,926,644	8,977,616
nvz without sorting	13,945,434	9,066,545
nvz **with** sorting	**12,833,126**	**8,869,852**
value of the sorting	**-8.0 %**	**-2.2 %**

That is the answer, and it concedes the objection: the value of the sorting is a function of the window and of how scattered the similarity was to begin with. With a small window and scattered similarity it is worth 8 %; with a dictionary covering the whole corpus, 2.2 %; on corpora that are already ordered, zero. nvz's measurable gains on the material in this report therefore come from the transforms, from deduplication where duplicates exist, and from nvcm in mode `liten` — **not** from the sorting.

```
bash experiment/ablation.sh kod xz9e
bash experiment/ablation.sh png zstd19
bash experiment/sortering.sh
```

---

8. Limitations and future work

Ordered by estimated return. Items 1–3 and 8–10 come from measurements made on 2026-11-18 prompted by questions in the public discussion of this work; they carry a measured value rather than a guess.

1. **Memory scales with the whole tree, not with the block.** Measured on Linux kernel 6.12 (1.48 GiB in 86,618 files): peak memory 2.1 GiB in mode `normal` and 3.3 GiB in `liten` — 1.4 and 2.2 times the raw data. The Python implementation streamed each block to a temporary file; the C++ rewrite instead holds each block's raw bytes in `Bygge::ra` until the block is compressed. The largest tree nvz can pack is therefore roughly available RAM divided by 1.4, and less for PNG-heavy trees, since an unfolded PNG is allowed to be up to 150 times its file size (`PNG_UPPBLASNING_TAK`). Remedy: keep `Bygge::ra` in a write-through temporary file and stream it back in step 5. This removes the tool's hardest limit and is the only item on this list that changes what nvz can handle at all.

1. **The default mode is the most extreme one.** `liten` is the default. Measured (3.2): where `liten` wins on size it costs 50–100 times longer unpacking — code 68 s vs. 1 s for a 12 % smaller archive, logs 203 s vs. 2 s for 17 %, enwik8 135 s vs. 2 s for 14 %, Silesia 101 s vs. 2 s for 6 %. Where it wins nothing it also costs nothing: on png, foto, gif, film and arkiv, `liten` and `normal` are equal in both size and time, because the codec probe rejects nvcm on its own. The trade-off is therefore real and works as intended; the objection is only that the default makes the choice for the user. Worth considering: `normal` as the default, with `liten` as an explicit choice.

1. **The small-file floor — and it is the register, not the compression.** An empty archive is 175 bytes (gzip: 25), and below roughly 30 kB of input nvz is larger than `xz -9e`: +14.2 % at 1 kB, +2.9 % at 10 kB, -11.8 % at 100 kB, -14.6 % at 1 MiB (the same text file, growing prefix). Breaking down a 1 kB case shows where the bytes are:

| | bytes | |:--|--:| | nvz, whole archive | 791 | | of which compressed payload | 616 | | of which register and tail | 175 | | `xz -9e`, whole file | 696 | | of which xz's container (header, block header, index, footer) | ~60 |

nvz's **payload is 80 bytes smaller than xz's entire file** — it is the register that loses the race, at three times xz's container. The fat entries are the object hash (16 raw bytes, and with a single object there is nothing to deduplicate against), the timestamp as an ISO string (~20 bytes), class and codec names as strings rather than numbers (~17), and the 16-byte tail holding two 64-bit values. Remedy: a compact register for single-file archives. That requires a v3 format, making it a format decision rather than a code fix: beating xz at 1 kB needs the register down to about 60 bytes, and the items listed give roughly half.

1. **The sample's blind spot — measured 2026-11-18, partly remedied.** The probe takes 4 MiB as eight disjoint pieces of the block. The construction is right for seeing what the data *is* (2.2), but it tears apart the long-range structure that distinguishes the codecs: eight pieces from separate parts of the block hold no repeats between them. On a 120 MiB block from the archive corpus the probe chose `zstd19` and got 100.13 MiB, while xz with a covering dictionary gave 91.97 — **the probe missed 8.2 %**.

A cheap predictive signal was tried and rejected: `zstd -3`'s ratio on the sample against the same ratio on the whole block does differ (1.016 and 1.025 on two blocks) but ranks them wrongly, so it cannot replace a measurement. The remedy was therefore to recompress the whole block with xz in mode `liten` when the probe chose something else — but only when the probe's margin was thin (below 8 %), since unconditional verification took enwik8 from 131 to 259 seconds without saving a byte.

What remains: `normal` still trusts the probe, and the 8 % threshold rests on two data points. A sample that grows with block size is still unmeasured.

1. **nvjpg sits below known technique.** -15.1 % against published figures of -20 to -22 % for packJPG, Lepton and brunsli (7). The gap is a modelling gap: Lepton models neighbouring coefficients in two dimensions and across block boundaries, nvjpg in one. Progressive JPEG is also left raw.

1. **PNG coverage stops at zlib.** Streams written by zlib-ng, libdeflate or Chrome cannot be reconstructed by nvz's recipe search. preflate solves the general case (7); linking it in widens coverage without touching the format.

1. **GIF recompression** (4.4): the same construction as nvjpg, applied to the LZW layer.

1. **Chunk-level instead of file-level deduplication.** File level finds nothing in rotated logs, disk images and successive versions of the same file, where content is shifted. Content-defined chunking has been known since LBFS (2001) and is used by borg and restic.

1. **Block size was a compile-time constant** (`BLOCKTAK`, 192 MiB) and is now the flag `--blocktak MiB`. The number of blocks governs both parallelism (one block is compressed by one thread, and on 24 threads half the machine idles when a tree yields eleven blocks) and the ratio (a block is the compressor's window). Measured on the PNG corpus in mode `normal`:

| block cap | pack time | archive | vs. 192 MiB | |--:|--:|--:|--:| | 192 MiB (default) | 261 s | 218,956,236 | ±0 | | 64 MiB | 201 s | 221,495,996 | **1.30x faster, +1.16 %** | | 32 MiB | 137 s | 231,569,236 | **1.90x faster, +5.76 %** |

The trade-off is therefore real and mildly convex: the first steps down buy a lot of time for almost no size, after which size gets expensive. The exception rule for opaque blocks (4.2) remains and is unaffected by the flag. Whether the default should change is unmeasured — it depends on what one optimises for — but the choice can now be made without a rebuild.

1. **Portability.** POSIX dependencies in the C++ part: `fork`, `pipe2`, `/proc/self/exe` (four sites), `lstat` (seven), `utimensat`, `dirent`, `symlink`. macOS requires `_NSGetExecutablePath` plus `pipe` with `fcntl`; Windows additionally requires the worker model to trade `fork`/`exec` for `CreateProcess` or threads, and a decision about symlinks and permissions. Today: Linux, or WSL2 on Windows.

1. **nvcm on opaque blocks:** mode `liten` sends only text classes to nvcm; whether opaque data pays off is unmeasured.

1. **AVX-512 is present in the machine but unused.** nvcm's inner loop is a dot product over ten int16 weights per bit plus ten table lookups. VNNI could take the dot product, but the lookups are scattered and are likely what dominates; this item is last because the gain is unmeasured and may be zero.

1. **The corpus is hard-linked to live working trees** (5.4). The fingerprint in `matning/matning.py` now detects when it moves, but the right remedy is to freeze it: `python3 korpus/bygg-korpus.py --kopiera`, followed by a re-measurement of the whole matrix. Until then every figure measured before 2026-11-18 carries that uncertainty.
2. **The memory budget is not deterministic.** The dictionary size — and with it both ratio and pack time — depends on how much memory the machine has free when nvz starts (4.5). That makes measurements dependent on other load. Remedy: set the budget to a share of the machine's TOTAL memory rather than its free memory, and state the share in the report.

Outside the evaluation: git packfiles (the same recipe approach as PNG, untested) and the GPU mode — measured separately, with the conclusion that LZ77 matching is accelerated 2.5 times but that serial entropy coding bounds the whole (Amdahl; see README).

Appendix A: Reproduction

Source code and all measurement data are published at <https://github.com/hugowikstrom/nvz>. Dependencies: g++ (C++17), libzstd-dev, liblzma-dev, zlib1g-dev; for the reference measurements also gzip, pigz, bzip2, zstd, and xz.

```
git clone https://github.com/hugowikstrom/nvz && cd nvz
make                                            # builds nvz, nvcm, nvjpg
python3 korpus/bygg-korpus.py gif film arkiv logg     # local corpora
export NVPACK_SKRAP=$PWD/.skrap
python3 matning/matning.py <corpus>                   # archive level, per corpus
python3 matning/standardkorpus.py silesia enwik8      # codec level, one thread
python3 prov/aterskapning.py gif arkiv logg film png foto kod
bash prov/prova-allt.sh                               # edge cases, 44 tests
python3 prov/slumpprov.py 400                         # randomised trees, 400 rounds
bash experiment/karntest.sh                           # Linux kernel, md5 before/after (3.7)
bash experiment/corpora.sh                            # zlib-ng/corpora, eight corpora (3.8)
bash experiment/ablation.sh kod xz9e                  # one trick at a time (7.1)
bash experiment/ablation.sh png zstd19
bash experiment/sortering.sh                          # sorting vs. window size (7.1)
bash experiment/arkiv-fonster.sh                      # hypothesis test of 4.2
python3 experiment/dubbletter.py
python3 rapport/bred-provning.py                      # fills the tables here
```