nvz — bred provning

Empirisk utvärdering av en innehållsmedveten arkivpackare: kompressionskvot, tidsåtgång och korrekthet över åtta datatyper och två standardkorpusar.

2026-11-16 · Ryzen AI 9 HX 470 (24 trådar), 28 GiB RAM, NVMe · källkod, mätdata och denna rapport: <https://github.com/hugowikstrom/nvz> · alla tal genereras ur `resultat/*.json` och kan reproduceras enligt appendix A

Sammanfattning

nvz är en arkivpackare som klassificerar sitt innehåll och tillämpar innehållsspecifika, förlustfria omvandlingar — deduplicering, likhetssortering, PNG-utveckling med receptåtersökning, JPEG-omkodning till aritmetisk kod samt kontextblandande kompression (nvcm) — innan den väljer kodek per block genom stickprov. Denna rapport utvärderar nvz mot sju etablerade packare på elva korpusar och prövar två hypoteser:

- **H1 (kompression):** innehållsmedvetenhet ger mätbart mindre arkiv än

generisk strömkompression på data med utnyttjbar struktur, utan att försämra resultatet på data som saknar sådan.

- **H2 (korrekthet):** varje arkiv återger källträdet bitexakt — innehåll,

rättigheter, tidsstämplar, symboliska länkar — och ett skadat arkiv avvisas i stället för att ge felaktiga data.

Resultaten stödjer båda hypoteserna med en kvantifierad avvikelse från H1. I sammandrag:

observation	mätvärde	referens
loggfiler, kvot	**23,8×** (10,03 av 238,3 MiB)	xz −9e: 20,1×
loggfiler, relativt bästa klassiska	**−15,3 %**	mot xz −9e
Silesia, nvcm (en tråd)	**1,677 bit/byte**	xz −9e: 1,829
enwik8, nvcm (en tråd)	**1,713 bit/byte**	xz −9e: 1,986
film, samtliga packare	inom 1 % av råstorlek	väntat: redan kodad
arkiv-av-arkiv, nvz mot zstd −22	**−0,1 %** efter åtgärd (+20,0 % före)	felsökt och rättat, avsnitt 4.2
återskapning, 7 korpusar	bitexakt, samtliga poster	avsnitt 5
skadeprov, 5 skadetyper	samtliga avvisade	avsnitt 5.2

Den ursprungligen uppmätta avvikelsen från H1 (+20,0 % på korpusen av redan komprimerade arkiv) felsöktes till kodekprovets stickprovsurval, åtgärdades under provningen och ommättes till −0,1 % mot zstd −22. En kompletterande mätning mot LZMA2:s maxform (7-Zip, 512 MiB ordbok; avsnitt 3.6) flyttar dock referensen: där ligger nvz +2,3 % efter på just den korpusen, medan försprånget på text snarare växer. H1 står därmed med en känd, kvantifierad motinstans — arkiv-av-arkiv mot LZMA2-extrem — och utan motinstans i övrigt.

---

1. Inledning

En konventionell arkivkedja (`tar | zstd`) behandlar trädet som en oordnad byteström: komprimeraren ser varken fildubbletter utanför sitt fönster, inbäddade komprimeringslager (deflate i PNG, Huffman i JPEG) eller möjligheten att placera likartade filer intill varandra. nvz utnyttjar samtliga tre, och varje omvandling

verifieras vid packning genom fullständig återsyntes och jämförelse mot originalet; en omvandling som inte kan bevisas reversibel faller tillbaka till rålagring.

Tidigare mätningar omfattade tre renodlade korpusar (programkod, PNG, JPEG-foton) och ett blandat projektträd. De besvarar inte hur packaren beter sig på datatyper utan utnyttjbar struktur (video, GIF, redan packade arkiv), på loggdata med hög redundans, eller i jämförelse med publicerade resultat. Föreliggande provning utvidgar underlaget med fyra korpusar av verklig data, två standardkorpusar (Silesia, enwik8), samt en systematisk korrekthetsprövning.

---

2. Metod

2.1 Jämförelsepackare

Sju konfigurationer av fyra etablerade verktyg: gzip (−6, −9 via pigz), bzip2 (−9), zstd (−3, −19 `--long=27`, −22 `--ultra --long=31`) och xz (−9e). `tar | zstd −3` betecknas *dagens* och utgör referensnivå, då den motsvarar maskinens befintliga arkivskript. nvz mäts i lägena `snabb` (zstd −3, inga omvandlingar), `normal` (omvandlingar, kodekprov per block) och `liten` (därtill nvcm som kandidat; standardläge), samt som `utan knep` (läge normal med deduplicering och omvandlingar avslagna) för att isolera omvandlingarnas bidrag.

2.2 Mätförfarande

Varje mätpunkt är en fullständig cykel: packning, uppackning till tom katalog, rekursiv jämförelse mot källträdet (`diff -rq --no-dereference`). Kolumnen *exakt* i tabellerna är utfallet av denna jämförelse, inte ett antagande. Tider är väggklocketid; packning får använda samtliga 24 trådar. Mätningarna kördes med `nice 19` och `ionice -c 3` på en i övrigt lastad maskin med lediga kärnor; kontrollmätningar visade ingen mätbar påverkan av prioriteten. Kodeknivåmätningen (avsnitt 3.3) följer i stället fackets konvention: en fil i taget, en tråd, ingen tar och ingen deduplicering.

Arkivstorleken är inte deterministisk mellan körningar: två identiska `--läge normal`-körningar gav 8 597 884 respektive 8 597 603 byte (skillnad $3{\cdot}10^{-5}$). Differenser inom några hundra byte är därför inte signifikanta.

2.3 Korpusar

De sex nedan kompletterar de tidigare tre (kod 55,3 MiB/5 974 filer, PNG 431,1/4 618, foto 408,5/1 143). Lokala korpusar byggs som hårda länkar av `korpus/bygg-korpus.py`; loggkorpusen kopieras i stället en gång, eftersom loggfiler skrivs under mätningen och en föränderlig korpus ogiltigförklarar cykeljämförelsen. Filer ändrade de senaste 15 minuterna utesluts.

korpus	omfång	innehåll	motivering
`gif`	105 filer, 27,4 MiB	webbgrafik, animationer, dubbletter i flera `node_modules`	LZW-kodat entropilager som nvz inte öppnar; hög dubblettandel
`film`	67 filer, 536,7 MiB	h264 (TikTok), VP9 (skärminspelning)	redan transformkodad data; förväntat nollresultat
`arkiv`	133 filer, 391,8 MiB	zip och tar.gz, varav två bitidentiska par och åtta dagliga säkerhetskopior av samma databas	komprimerad data med redundans *mellan* filer
`logg`	76 filer, 238,3 MiB	syslog, kern.log, apploggar, JSONL-transkript	maximalt repetitiv text; växande arkivtyp
`silesia`	12 filer, 211 938 580 B	Silesia-korpusen	jämförbarhet: standard i lzbench m.fl.

korpus	omfång	innehåll	motivering
`enwik8`	1 fil, 10^8 B	engelska Wikipedia, första 10^8 byte	jämförbarhet: Large Text Compression Benchmark

Dubblettandelen per korpus kvantifieras i tabellen nedan, eftersom den avgör hur stor del av en kvot som förklaras av deduplicering snarare än kompression.

korpus	filer	unika filer	rå MiB	exakta dubbletter MiB	andel
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. Resultat

3.1 Översikt

Tabellen sammanfattar bästa nvz-läge mot referensnivån och mot det bästa klassiska verktyget per korpus. Negativa värden anger mindre arkiv.

korpus	rå MiB	dagens MiB	bästa klassiska MiB	bästa nvz MiB	mot dagens	mot bästa klassiska	packtid nvz/klassisk
Programkod	55	11,5	8,2 (xz -9e)	7,2 (liten)	−37,1 %	−12,5 %	68 / 47 s
Bilder (PNG)	431	362,9	245,2 (zst -22)	207,2 (liten)	−42,9 %	−15,5 %	280 / 156 s
Foton (JPEG)	409	406,3	404,9 (bz2 -9)	345,1 (normal)	−15,1 %	−14,8 %	48 / 57 s
Bilder (GIF)	27	27,0	14,2 (zst -22)	14,2 (normal)	−47,6 %	−0,0 %	6 / 5 s
Film (h264 och VP9)	537	533,9	532,0 (zst -19)	532,0 (normal)	−0,3 %	+0,0 %	56 / 57 s
Arkiv (zip och tar.gz)	392	381,8	242,8 (zst -22)	237,0 (liten)	−37,9 %	−2,4 %	323 / 154 s
Loggfiler	238	22,4	11,8 (xz -9e)	10,0 (liten)	−55,2 %	−15,4 %	191 / 897 s
Silesia (arkivnivå)	202	63,1	46,2 (xz -9e)	43,5 (liten)	−31,0 %	−5,7 %	100 / 158 s
enwik8 (arkivnivå)	95	33,8	23,7 (xz -9e)	20,4 (liten)	−39,6 %	−13,7 %	129 / 121 s

3.2 Hela matrisen

Samtliga packare över samtliga korpusar, i tre paneler: storlek, packtid, uppacktid. Kolumnerna silesia och enwik8 avser arkivnivån (hela korpusen genom tar-ekvivalent väg), inte kodeknivån i 3.4.

Arkivstorlek

packare	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 korpus; minsta värde per kolumn i fet. Råstorlekar: kod 55, png 431, foto 409, gif 27, film 537, arkiv 392, logg 238, silesia 202, enwik8 95 MiB.

Packtid

packare	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 korpus; minsta värde per kolumn i fet. Råstorlekar: kod 55, png 431, foto 409, gif 27, film 537, arkiv 392, logg 238, silesia 202, enwik8 95 MiB.

Uppacktid

packare	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
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 korpus; minsta värde per kolumn i fet. Råstorlekar: kod 55, png 431, foto 409, gif 27, film 537, arkiv 392, logg 238, silesia 202, enwik8 95 MiB.

3.3 Per korpus

Fullständiga serier per korpus. *Mot dagens* anger storleksförändring relativt `tar.zst -3`.

Bilder (GIF) — 27.4 MiB i 105 filer

packare	storlek MiB	kvot	mot dagens	packtid	uppacktid	exakt
tar (okomprimerad)	27,62	0,99×	+2,1 %	0 s	0 s	ja
tar.gz -6	27,18	1,01×	+0,5 %	1 s	0 s	ja
tar.gz -9 (pigz)	27,19	1,01×	+0,6 %	0 s	0 s	ja
tar.bz2 -9	27,28	1,00×	+0,9 %	4 s	3 s	ja
tar.zst -3 (dagens)	27,04	1,01×	+0,0 %	**0 s**	0 s	ja
tar.zst -19 --long	14,18	1,93×	−47,6 %	4 s	0 s	ja
tar.zst -22 --ultra	14,18	1,93×	−47,6 %	5 s	0 s	ja
tar.xz -9e	14,19	1,93×	−47,5 %	8 s	0 s	ja
nvz snabb	14,44	1,90×	−46,6 %	0 s	0 s	ja
nvz utan knep	14,18	1,93×	−47,6 %	4 s	0 s	ja
nvz normal	**14,18**	1,93×	−47,6 %	6 s	0 s	ja
nvz liten	**14,18**	1,93×	−47,6 %	6 s	0 s	ja

Film (h264 och VP9) — 536.7 MiB i 67 filer

packare	storlek MiB	kvot	mot dagens	packtid	uppacktid	exakt
tar (okomprimerad)	536,82	1,00×	+0,6 %	1 s	0 s	ja
tar.gz -6	532,39	1,01×	−0,3 %	24 s	2 s	ja

packare	storlek MiB	kvot	mot dagens	packtid	uppacktid	exakt
tar.gz -9 (pigz)	532,54	1,01×	−0,2 %	2 s	2 s	ja
tar.bz2 -9	535,53	1,00×	+0,3 %	78 s	54 s	ja
tar.zst -3 (dagens)	533,86	1,01×	+0,0 %	**0 s**	0 s	ja
tar.zst -19 --long	**532,02**	1,01×	−0,3 %	57 s	0 s	ja
tar.zst -22 --ultra	532,03	1,01×	−0,3 %	266 s	1 s	ja
tar.xz -9e	532,86	1,01×	−0,2 %	154 s	5 s	ja
nvz snabb	536,75	1,00×	+0,5 %	3 s	1 s	ja
nvz utan knep	532,04	1,01×	−0,3 %	57 s	1 s	ja
nvz normal	532,04	1,01×	−0,3 %	56 s	1 s	ja
nvz liten	532,04	1,01×	−0,3 %	57 s	1 s	ja

Arkiv (zip och tar.gz) — 391.8 MiB i 133 filer

packare	storlek MiB	kvot	mot dagens	packtid	uppacktid	exakt
tar (okomprimerad)	391,93	1,00×	+2,7 %	1 s	0 s	ja
tar.gz -6	380,05	1,03×	−0,5 %	18 s	2 s	ja
tar.gz -9 (pigz)	380,14	1,03×	−0,4 %	1 s	2 s	ja
tar.bz2 -9	384,49	1,02×	+0,7 %	56 s	39 s	ja
tar.zst -3 (dagens)	381,78	1,03×	+0,0 %	**0 s**	1 s	ja
tar.zst -19 --long	261,37	1,50×	−31,5 %	47 s	0 s	ja
tar.zst -22 --ultra	242,77	1,61×	−36,4 %	154 s	1 s	ja
tar.xz -9e	251,75	1,56×	−34,1 %	144 s	7 s	ja
nvz snabb	329,03	1,19×	−13,8 %	2 s	1 s	ja
nvz utan knep	242,51	1,62×	−36,5 %	146 s	1 s	ja
nvz normal	242,51	1,62×	−36,5 %	153 s	1 s	ja
nvz liten	**237,00**	1,65×	−37,9 %	323 s	8 s	ja

Loggfiler — 238.3 MiB i 76 filer

packare	storlek MiB	kvot	mot dagens	packtid	uppacktid	exakt
tar (okomprimerad)	238,39	1,00×	+965,0 %	0 s	0 s	ja
tar.gz -6	30,46	7,82×	+36,1 %	4 s	1 s	ja
tar.gz -9 (pigz)	29,03	8,21×	+29,7 %	1 s	1 s	ja
tar.bz2 -9	18,92	12,60×	−15,5 %	32 s	6 s	ja
tar.zst -3 (dagens)	22,38	10,65×	+0,0 %	**0 s**	0 s	ja
tar.zst -19 --long	13,09	18,21×	−41,5 %	113 s	0 s	ja
tar.zst -22 --ultra	12,92	18,45×	−42,3 %	986 s	0 s	ja
tar.xz -9e	11,84	20,12×	−47,1 %	897 s	1 s	ja
nvz snabb	22,41	10,63×	+0,1 %	2 s	1 s	ja

packare	storlek MiB	kvot	mot dagens	packtid	uppacktid	exakt
nvz utan knep	11,87	20,08×	−47,0 %	854 s	2 s	ja
nvz normal	11,87	20,08×	−47,0 %	851 s	2 s	ja
nvz liten	**10,03**	23,77×	−55,2 %	191 s	198 s	ja

3.4 Standardkorpusar, kodeknivå

Jämförelsen avser kompressorn nvcm isolerad — en fil i taget, en tråd — eftersom publicerade tal för dessa korpusar är definierade så. nvz som arkivpackare ingår i 3.1–3.3.

Silesia — 211 938 580 byte

kodek	storlek byte	kvot	bits/byte	pack MiB/s	upp MiB/s	exakt
gzip -9	67 631 990	3,134×	2,553	6,8	157,9	ja
bzip2 -9	54 506 769	3,888×	2,058	8,1	19,2	ja
zstd -19	52 895 350	4,007×	1,997	1,6	374,3	ja
zstd -22 --ultra	52 369 178	4,047×	1,977	1,2	360,9	ja
brotli -q 11	49 564 563	4,276×	1,871	0,3	175,8	ja
xz -9e	48 456 004	4,374×	1,829	1,3	66,5	ja
nvcm	44 439 240	4,769×	1,677	0,9	0,8	ja

enwik8 — 100 000 000 byte

kodek	storlek byte	kvot	bits/byte	pack MiB/s	upp MiB/s	exakt
gzip -9	36 445 248	2,744×	2,916	10,2	138,2	ja
bzip2 -9	29 008 758	3,447×	2,321	8,7	16,5	ja
zstd -19	26 944 227	3,711×	2,155	1,1	340,6	ja
brotli -q 11	25 742 001	3,885×	2,059	0,3	207,3	ja
zstd -22 --ultra	25 272 471	3,957×	2,022	0,8	381,5	ja
xz -9e	24 831 648	4,027×	1,986	0,8	69,6	ja
nvcm	21 417 184	4,669×	1,713	0,8	0,7	ja

Per fil på Silesia, där korpusens heterogenitet är själva poängen:

fil	innehåll	rå byte	zstd -19	brotli -q 11	xz -9e	nvcm	nvcm mot bäst
`dickens`	engelsk litteratur, text	10 193 352	2 849 945	2 827 777	2 831 212	2 234 894	−20,2 %
`mozilla`	körbara filer (tar)	51 215 653	15 071 672	13 872 265	13 376 240	14 366 242	+7,4 %
`mr`	medicinsk 3D-bild	9 970 424	3 106 737	2 823 136	2 751 892	2 205 358	−9,7 %
`nci`	kemiska strukturer, text	33 553 317	1 664 988	1 519 768	1 449 272	1 505 038	+3,8 %
`ooffice`	körbar DLL	6 151 755	2 595 377	2 478 855	2 427 224	2 379 789	−2,0 %
`osdb`	databasdump, binär	10 084 646	3 100 177	2 816 278	2 844 556	2 395 972	−14,5 %
`reymont`	polsk litteratur, PDF-källa	6 627 493	1 348 462	1 332 158	1 315 592	1 053 488	−15,5 %
`samba`	källkod (tar)	21 606 321	3 894 675	3 766 340	3 739 524	3 377 043	−9,7 %

fil	innehåll	rå byte	zstd -19	brotli -q 11	xz -9e	nvcm	nvcm mot bäst
`sao`	stjärnkatalog, binär	7 250 892	5 000 576	4 586 092	4 425 664	4 509 261	+1,9 %
`webster`	ordbok, text	41 458 740	8 679 705	8 428 575	8 368 672	6 317 041	−24,5 %
`x-ray`	röntgenbild, 16-bitars	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 %

Som referenspunkt mot publicerade resultat (LTCB, andra maskiner och programversioner; ej egna mätningar):

program	enwik8 (byte)	bit/byte	metod
gzip 1.3.5	36 445 248	2,92	ordbok
bzip2 1.0.2	29 008 736	2,32	blocksortering
zstd 0.6.0	25 405 601	2,03	ordbok
xz 5.2.1	24 703 772	1,98	ordbok (LZMA)
nanozip 0.09a	18 594 163	1,49	kontextblandning
zpaq 6.42	17 855 729	1,43	kontextblandning
paq8px_v206fix1	15 849 084	1,27	kontextblandning
cmix v21	14 623 723	1,17	kontextblandning

Vår egen mätning: **nvcm gör enwik8 på 21 417 184 byte, 1,713 bits/byte** — 13,8 % mindre än `xz -9e` på samma maskin (24 831 648 byte, 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 Genomströmning

Sekunder är inte jämförbara mellan korpusar av olika storlek; tabellen anger därför MiB/s rådata genom väggklockan.

korpus	nvz normal pack	nvz normal upp	zstd -3 pack	zstd -19 pack	xz -9e pack
Programkod	1	97	326	2	1
Bilder (PNG)	3	51	743	11	3
Foton (JPEG)	9	23	869	8	3
Bilder (GIF)	5	304	304	6	3
Film (h264 och VP9)	10	378	1 220	9	3
Arkiv (zip och tar.gz)	3	353	1 059	8	3
Loggfiler	0	125	1 254	2	0
Silesia (arkivnivå)	2	82	612	3	1
enwik8 (arkivnivå)	1	49	502	2	1

MiB per sekund, rådata genom väggklockan.

3.6 LZMA2 i maxform

`xz -9e` i tabellerna ovan är LZMA2 med 64 MiB ordbok. Formatets maxform är en annan motståndare: 7-Zip med `-mx=9`, **512 MiB ordbok** och solid arkiv — ordboken täcker då hela varje korpus utom film, samma spelplan som nvz:s stora block. Jämförelsen kördes som hel rundgång (`experiment/lzma2-extrem.sh`), inga filter, enbart LZMA2 mot nvz i standardläget:

korpus	rå MiB	LZMA2-extrem MiB	nvz MiB	nvz mot LZMA2-extrem	packtid LZMA2/nvz	exakt
Programkod	55	8,22	**7,28**	−11,4 %	19 / 70 s	ja
Loggfiler	238	14,16	**10,03**	−29,2 %	70 / 196 s	ja
Arkiv (zip/tar.gz)	392	237,12	242,53	+2,3 %	43 / 195 s	ja
Bilder (PNG)	431	242,99	**208,81**	−14,1 %	40 / 268 s	ja
Silesia	202	46,43	**43,54**	−6,2 %	78 / 102 s	ja
enwik8	95	23,65	**20,43**	−13,6 %	79 / 132 s	ja

Två observationer. nvz:s försprång på text består mot även den starkaste LZMA2-formen — kontextblandningen tar hem det en större ordbok inte kan. På arkivkorpusen var LZMA2-maxformen däremot starkare än både zstd −22 och nvz, och därmed den nya bästa klassiska referensen där: **nvz låg +2,3 % efter**. 7-Zip är dessutom genomgående snabbare i packning (flertrådad LZMA2).

Åtgärdat 2026-11-18. Avståndet hade en mekanism och inte bara en siffra: nvz:s LZMA-kodek körde `preset 9e`:s fasta ordbok på 64 MiB, medan 7-Zips maxform hade 512 MiB och täckte därmed hela blocket. nvz:s ordbok skalas nu efter blocket (se 4.2 och kommentaren i `src/cpp/kodek.cpp`), och LZMA fick plats som kandidat i den opaka eskaleringen. Ommätt ligger nvz 0,05 % **under** LZMA2-maxformen på arkivkorpusen — 248 512 199 mot 248 636 548 byte. Raden i tabellen ovan är därmed inaktuell för arkiv; den står kvar som den mättes 2026-11-17, och ommätningen står i 4.2.

3.7 Linuxkärnan: en korpus som inte är vår

Samtliga korpusar ovan är valda av oss, och invändningen att ett resultat kan vara anpassat till sin testdata kan inte besvaras med mer av samma data. `experiment/karntest.sh` kör därför hela rundgången på ett träd från tredje part: linuxkärnan 6.12, hämtad från `cdn.kernel.org`, med en checksumma vi inte satt själva. Ingen tröskel och ingen parameter i nvz har valts med den i sikte.

packare	byte	mot kernel.orgs egen	packtid
kernel.org `tar.xz` (som den levereras)	147 906 904	±0	—
`tar.zst -19 --long`	147 274 821	−0,4 %	117 s
`7z` LZMA2 `-mx=9`, 512 MiB ordbok, solid	141 350 334	−4,4 %	264 s
`tar.xz -9e` (lokalt, 64 MiB ordbok)	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 byte i 86 618 filer. Uppackning `liten`: 439 s.

Mot bästa klassiska verktyg på samma maskin är nvz 12,0 % mindre, och mot LZMA2 i maxform (3.6) 12,8 % mindre. `normal` ligger 1,2 % efter `xz -9e` men packar 1,5 gånger snabbare. Att utfallet (−12,0 %) ligger i samma storleksordning som den egna kodkorpusen (−11,6 %, 3.1) är provets egentliga resultat: försprånget följer med till data som inte valts av oss.

Korrekthetsdelen kördes som manifest före och efter, kontrollerad av `md5sum -c` och inte av nvz egen kod: **86 618 av 86 618 filer OK, 0 FAILED**, och manifesten identiska rad för rad.

3.8 zlib-ng/corpora: åtta korpusar valda av någon annan

Förslaget kom utifrån, och värdet ligger i urvalet: `zlib-ng/corpora` samlar litteraturens gamla standardkorpusar — Calgary och Canterbury — jämte `artificial` (patologiska fall), `neuro` (mätdata), `snappy`, `large` och Silesia. Calgary och Canterbury består av **små** filer, vilket är det område där nvz är svagast (avsnitt 8, punkt 3), så urvalet är inte gynnsamt valt. Samtliga rader är provade med full rundgång och bitexakt jämförelse.

korpus	rå	gzip –9	bzip2 –9	zstd –19	xz –9e	nvz `normal`	nvz `liten`	nvz mot bästa klassiska
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 %**

Byte per korpus, hela underkatalogen som ett träd. Rundgången bitexakt i samtliga 24 körningar (tre lägen × åtta korpusar).

Sju av åtta korpusar går till nvz, med −15,8 % på Canterbury som bästa utfall. Den åttonde är `neuro`, där nvz ligger **1,0 % efter `xz -9e`**: materialet är binär mätdata utan de likheter mellan filer som nvz:s omvandlingar och dedup lever av, och då återstår bara kodeken — och där är LZMA med 64 MiB ordbok starkare än nvcm på just den datan. Det är samma mönster som arkivkorpusen i 4.2 och ska läsas som gränsen för H1: nvz förbättrar data med utnyttjbar struktur, och på data utan sådan är resultatet en knapp procent sämre, inte bättre.

3.9 Dominans: bättre OCH snabbare, korpus för korpus

Tabellerna ovan jämför storlek och tid var för sig. Den hårdare frågan är om nvz kan slå **samma** motståndare på båda axlarna samtidigt — ett krav som restes 2026-11-18: *bättre än det etablerade, överallt, snabbare och minst lika bra.* `experiment/dominans.sh` prövar det: motståndaren per korpus är den av de etablerade packarna som gav minsta arkiv, med sin egen uppmätta packtid på samma maskin, och av nvz:s lägen räknas det minsta av dem som dominerar.

```
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
```

Mönstret är inte slumpmässigt. nvz dominerar där det finns struktur att utnyttja — källkod, loggar, foton, bilder, blandade träd — och faller på TID, aldrig på storlek, där det inte finns något att utnyttja: film och gif är redan entropikodade, arkivkorpusen kräver en hel extra LZMA-packning för de sista procenten, och enwik8 är en enda textfil där varken dedup, sortering eller omvandlingar har något att bidra med och nvz:s fem svep över datan (README, "Golvet är inte kodeken") blir ren kostnad.

Gränsen går alltså inte mellan datatyper utan mellan *utnyttjbar struktur och ingen*: samma slutsats som H1, men mätt på två axlar i stället för en.

---

4. Analys

4.1 Loggdata: kompressorvalet dominerar

Loggkorpusen uppvisar provningens högsta kvot, 23,8×, och källan kan isoleras genom lägesjämförelsen: `normal` och `liten` skiljer sig enbart i att nvcm deltar i kodekprovet.

läge	storlek	packtid	aktiv skillnad
`normal`	11,97 MiB	864 s	kodekprov zstd/xz
`liten`	**10,03 MiB**	**197 s**	därtill nvcm

nvcm vann provet för samtliga textblock, vilket arkivregistret bekräftar (2 block, båda nvcm; 192,03 → 9,00 MiB respektive 46,29 → 1,03 MiB). Att det mindre arkivet också packas snabbare beror på parallellitet: nvz:s block komprimeras oberoende av varandra, medan `xz -9e -T0` på denna datamängd i praktiken exekverar enkeltrådat till följd av sin minnesbudget per tråd. Blockens olika kvoter (21,4× respektive 45,1×) illustrerar adaptionens förlopp: modellen som sett 192 MiB logg predikterar nästa block väsentligt bättre.

4.2 Redan komprimerade arkiv: stickprovets gräns

På arkivkorpusen är nvz 20,0 % större än `zstd −22 --ultra --long=31` (291,3 mot 242,8 MiB). Två förklaringshypoteser prövades experimentellt (`experiment/arkiv-fonster.sh`): (a) blockpartitioneringen hindrar matchning mellan block; (b) kodekprovet väljer fel kodek.

konfiguration	storlek
zstd –19, standardfönster (8 MiB)	365,3 MiB
zstd –19 `--long=27`, hel ström	261,4 MiB
zstd –22 `--ultra --long=31`, hel ström	242,8 MiB
nvz, kodekprovet väljer	291,3 MiB
nvz, `--kodek zstd19`	260,2 MiB
nvz, `--kodek zstd22`	**244,6 MiB**

Hypotes (a) falsifieras: med manuellt vald kodek kostar partitioneringen 1,2 MiB (0,7 %). Hypotes (b) bekräftas: provet komprimerar 4 MiB spridda över blocket, och korpusens redundans ligger i nästan identiska filer hundratals megabyte isär — en egenskap ett glest stickprov inte kan observera. Provet klassificerar därför blocket som svårkomprimerat och väljer en billig kodek. Deduplikationen fångar de exakta kopiorna (62,8 MiB, 16,0 %); differensen utgörs av *nästan*-kopior (olika gzip-strömmar av likartad rådata), som endast långdistansmatchning över hela strömmen ser.

Åtgärd och ommätning. Diagnosen implementerades som två regler: (i) opaka block delas inte vid 192 MiB — på opak data är blockdelning en kompressionsgräns, inte en minnesoptimering, och zstd parallelliserar inom blocket; (ii) för opaka block som stickprovet klassar som komprimerbara ersätts stickprovsvalet av en tvåstegsregel, eftersom stickprovet inte kan skilja video från arkiv-av-arkiv — båda är lokalt täta, och skillnaden syns först över hela blocket. Blocket packas först med zstd −19 (fönster 2^{27}); vinner den mer än 10 % finns redundans mellan filer, och blocket packas om med zstd −22 (fönster 2^{31}) som tar hem resten. Video stannar vid någon procent och betalar aldrig för den dyra vägen. Ommätt resultat:

konfiguration	storlek
zstd –22 `--ultra --long=31`, hel ström	242,8 MiB
nvz efter åtgärd (1 block, eskalerad till zstd22)	**242,5 MiB**

Filmkorpusen prövar regelns andra gren: eskaleringen uteblir (zstd −19 vinner 0,9 %), och packtiden stannar vid 60 s mot de 255 s en ovillkorlig zstd −22 hade kostat.

nvz går därmed från +20,0 % till −0,1 % mot bästa klassiska verktyg på korpusen; differensen utgörs av deduplikationens 62,8 MiB, som långdistansmatchningen i det klassiska fallet måste återupptäcka som matchningar i stället för att aldrig lagra.

Andra ommätningen, 2026-11-18. Tvåstegsregeln hade två svagheter som mättes fram när frågan ställdes om nvz kan vara bättre än det bästa etablerade verktyget *överallt*:

Det billiga steget var inte billigt. Regeln använde `zstd −19` för att svara på frågan "har blocket tiotals procent att ge?". Men den frågan behöver ingen stark kompressor, bara en ärlig körning över hela blocket. Mätt på gif-korpusen:

kodek över hela blocket	storlek	tid
`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` ser 47,8 procentenheter av de 48 som finns att hämta, på en sextiondel av tiden. Första steget är därför `zstd −3`. Grenen för filmlik data behåller `zstd −19` som slutsteg, eftersom den både är billigare och bättre än `zstd −22` på sådan data (film: 557 863 118 byte på 56 s mot 557 873 791 på 264 s).

Den dyra grenen saknade sin starkaste kandidat. Där stickprovet visat att blocket har likheter mellan filer provas i läget `liten` numera även **LZMA med en ordbok som täcker hela blocket** — den enda kandidat vars fönster kan nå över ett opakt block på flera hundra megabyte, och exakt den form 7-Zip slog nvz med i 3.6. Med den:

konfiguration	storlek	mot nvz
`zstd -22 --ultra --long=31`, hel ström	254 310 751	+2,3 %
7-Zip LZMA2 `-mx=9`, 512 MiB ordbok, solid	248 636 548	+0,04 %
nvz `liten` efter åtgärd	**248 512 199**	—

nvz är därmed **0,04 % mindre än LZMA2 i maxform** på den korpus som var provningens enda motinstans mot H1. Priset är tid: 188 s mot 7-Zips 43 s, eftersom steget är en hel extra packning av blocket. Den avvägningen är inramad i lägena — `normal` gör den inte, `liten` gör den.

4.3 Video: nollresultatet och dess kostnad

Samtliga tolv konfigurationer landar inom 1 % av råstorleken — väntat, eftersom transformkodad video saknar entropilager åtkomligt för generisk kompression. Den relevanta jämförelsen är därför kostnaden för att konstatera detta: nvz `normal` når samma storlek som `zstd -22` på 16 % av tiden (61 mot 266 s), eftersom kodekprovet klassificerar blocken som rålagringsfall efter stickprov i stället för att komprimera dem fullt ut.

4.4 GIF: kvot utan kompressionsbidrag

GIF-korpusens kvot 1,93× förklaras i sin helhet av dubbletter (47,3 % av korpusen) och av att korpusen (27,4 MiB) ryms inom samtliga moderna packares fönster: nvz, zstd −19/−22 och xz når samma storlek, och `nvz utan knep` är identisk med `nvz normal`. nvz:s omvandlingar bidrar med noll på LZW-kodat innehåll — formatet öppnas inte. En GIF-motsvarighet till nvjpg (LZW → aritmetisk kodning av bildrutorna) är därmed en identifierad men omätt möjlighet; korpusen för att mäta den finns nu.

4.5 Ordboken, minnet och parallelliteten

Att låta LZMA:s ordbok täcka blocket (4.2) gav 2,17 % på arkivdata. Men ändringen hade en andra ordningens effekt som bara syns på en maskin med ont om minne, och den var värre än vinsten.

Kodaren behöver omkring tio gånger ordboken. PNG-korpusens block är 192 MiB och fick därmed en ordbok på 256 MiB och ett minnesbehov på 2,7 GiB styck. Nio sådana block ryms inte i ledigt minne, så grinden släppte fram **ett åt gången**: packtiden gick från 268 till 841 sekunder för 0,06 % mindre arkiv. En ordboksvinst som tar bort parallelliteten är ingen vinst.

Tre rättelser, i ordning efter vad de gav:

ändring	PNG-korpusens packtid
ordbok som täcker blocket, utan hänsyn till minnet	841 s
ordboken krymps tills den ryms i budgeten delad på antalet samtidiga kodare	191 s
kodekprovet görs per klass i stället för per block	178 s
ordboken täcker blocket men blir aldrig större, och budgeten är hela det lediga minnet	**129 s**

Den sista raden är den viktigaste och den mest självklara i efterhand: ett block på 32 MiB fick en ordbok på 64 MiB därför att golvet var satt till `preset 9e`:s värde. En ordbok som är större än blocket kan per konstruktion inte användas — men den kostar minne, och minnet är det som avgör hur många block som packas samtidigt. Golvet är därför borta: ordboken är exakt vad blocket behöver, avrundat uppåt till en tvåpotens.

Därtill delas klassen `png-ra` — utvecklade PNG-filterrader — vid 32 MiB i alla lägen utom `liten`. Klassen är särskild på tre sätt: den är den enda som blåses upp (431 MiB filer blir 2 GiB block), dess likheter ligger mellan närliggande bilder snarare än hundratals megabyte bort, och den är därmed den klass där blockdelning kostar minst kvot per sparad sekund:

blocktak för png-ra	packtid	arkiv
192 MiB (`liten`)	280 s	217 308 163
32 MiB (`normal`)	**127 s**	230 452 176

Bästa etablerade på korpusen är `zstd -22` med 257 132 369 byte på 156 s. `normal` är därmed 10,4 % mindre **och** 18 % snabbare, medan `liten` fortfarande tar de sista 5,7 procenten för den som vill ha dem. Det är avvägningen lägena finns för, nu mätt i båda ändar.

Priset för hela det här avsnittet står också skrivet: nvz:s ordbok och därmed både kvot och tid beror numera på hur mycket minne maskinen har ledigt. Mätningar ska därför göras på en maskin utan annan last — annars mäter man grinden och inte kompressionen.

---

5. Korrekthet

5.1 Återskapning

Rekursiv diff besvarar en svagare fråga än H2 kräver: den jämför innehåll i filer som existerar på båda sidor, men är blind för rättigheter, tidsstämplar, tomma kataloger och poster som tyst fallit bort. `prov/aterskapning.py` beräknar därför ett fullständigt avtryck — SHA-256, läge, mtime i nanosekunder, länkmål, posttyp — före packning och efter uppackning, kräver identitet, samt verifierar att arkivregistrets postantal överensstämmer med trädets.

korpus	filer	poster i registret	SHA-256 lika	läge och tider lika	hårda länkar kvar	avvikelser
gif	105	ja	ja	ja	nej	inga
arkiv	133	ja	ja	ja	nej	inga
logg	76	ja	ja	ja	nej	inga
film	67	ja	ja	ja	nej	inga
png	4 618	ja	ja	ja	nej	inga
foto	1 143	ja	ja	ja	nej	inga
kod	5 974	ja	ja	ja	nej	inga

Skadeprov på arkivet från korpusen **gif**:

skada	utfall	godkänt
magin bytt	vägrade	ja
byte i blockkropp	vägrade	ja
sista kilobyten bort	vägrade	ja
registerpekaren noll	vägrade	ja
en byte i registret	vägrade	ja

Därtill prövar `prov/prova-allt.sh` kantfall — tomma filer, dubbletter, relativa, brutna och katalogpekande symlänkar, filnamn med radbrytning och åäö, rättigheter 400–755, tidsstämplar 1985 och 2031 — i samtliga

lägen, med krav på full metadataidentitet. Sviten omfattar 44 prov och passerar utan anmärkning.

Kantfallssviten provar emellertid bara de fall någon har kommit på. `prov/slumpprov.py` provar kombinationer ingen kommit på: slumpade filstorlekar, filtyper, dubblettmönster, rättigheter, tider, namn och symlänkar, med verkliga PNG:er och JPEG:ar ur korpusen inblandade, och varje runda är en hel rundgång i samtliga lägen plus arkivets egen kontroll. **400 rundor (frön 5 000–5 399) kom tillbaka bitexakt, noll avvikelser.** En runda som misslyckas skriver ut sitt frö, så felet går att återskapa exakt — och det är den mekanismen som fällde byggfelet i avsnitt 6, punkt 4, på första rundan.

5.2 Skadetålighet

En packare som vid skada returnerar felaktiga data utan felindikation är värre än en som vägrar: den förra upptäcks först vid användning. Fem skadeklasser injicerades i ett friskt arkiv; kravet är avvisning eller bevisat oskadd utdata, aldrig tyst korruption.

skadeklass	utfall
utbytt magi	avvisat
bitfel i blockkropp	avvisat
trunkering (sista 1 024 byte)	avvisat
nollställd registerpekare	avvisat
bitfel i registret	avvisat

5.3 Versionskompatibilitet

C++-implementationen (avsnitt 6) läser Python-implementationens arkiv (JSON-register, 8 bytes hashprefix) bitexakt, inklusive PNG- och JPEG-omvandlade objekt; detta ingår i provsviten. Det omvända stöds inte: v2-registret är binärt, och Python-implementationen är utgången.

Registerversion 3 (2026-11-18). Registret bär sedan dess tre ändringar, alla gjorda för att arkivet skulle bli mindre än det bästa etablerade verktygets på filrika träd:

ändring	varför	vinst på kodkorpusen
objekthashen 16 → 4 byte	hashen används av `-t` för att upptäcka felmonterat innehåll, och 32 bitar räcker för det; bitfel fångas redan av blockens CRC64/xxhash. Hashar är slumpmässiga och därmed inkomprimerbara	70 776 byte
tiden som epoksekunder i stället för ISO-sträng	5 byte i stället för 20	15 byte
registret packas med den minsta av xz och zstd	registret är text och tal; xz tar det bättre, men dess container är ~60 byte mot zstds ~13 och förlorar därför på små arkiv	12 005 byte

Registret gick från 239 049 till 153 596 byte — från 2,76 % av arkivet till 1,79 % — och hela arkivet från 8 675 609 till 8 578 823 byte. **Läsningen är bakåtkompatibel:** v1 (JSON), v2 (16 bytes hash, zstd) och v3 läses alla av samma binär, och vilken kompressor registret packades med avgörs av strömmens egen magi och inte av versionsfältet, som ligger inuti det komprimerade registret. Det omvända gäller inte: en nvz byggd före 2026-11-18 kan inte läsa ett v3-arkiv. Provsviten täcker alla tre versionerna.

5.4 Mätningens egen felkälla: en levande korpus

Under ommätningen 2026-11-18 rapporterade två rader "SKILJER" på korpusen `kod` — den rekursiva jämförelsen fann tre filer som inte kom tillbaka likadana. Felet låg inte i nvz.

`korpus/bygg-korpus.py` **hårdlänkar** i stället för att kopiera, för att spara disk. Filen i `korpus/kod` är därför samma inod som filen i det levande arbetsträdet, och det gäller nästan allt:

korpus	filer med fler än en länk
kod	5 971 av 5 974
png	4 618 av 4 618
foto	1 143 av 1 143
gif	105 av 105
film	67 av 67
arkiv	131 av 133
logg	0 av 76

De tre filerna låg i `korpus/kod/ullabella/`, och deras mtime föll mellan packningen och jämförelsen: någon sparade i ett annat fönster medan mätningen pågick. Femton filer i korpusen hade ändrats under de närmast föregående två timmarna. Arkivet var alltså felfritt; det var korpusen som hade rört sig.

Loggkorpusen har alltid kopierats av precis detta skäl — det står i `bygg-korpus.py`:s eget huvud: *"uppackning gör varje jämförelse meningslös"*. Slutsatsen gäller alla korpusar, inte bara loggar. Två åtgärder:

1. `matning/matning.py` tar numera ett billigt avtryck av korpusen (sökväg, storlek och mtime per fil, inget innehåll läst) före och efter varje mätning, och skriver fältet `korpus_stabil` i resultatet. En mätning där korpusen rörde sig är ogiltig och ska göras om — inte tolkas.
2. `korpus/bygg-korpus.py --kopiera` bygger korpusen med riktiga kopior i stället för hårda länkar. Det kostar ungefär 2,3 GiB disk och gör korpusen stillastående.

De två raderna mättes om på en stabil korpus och är bitexakta. Men konsekvensen är bredare än två rader: **varje tal i den här rapporten som mättes före den här dagen vilar på en korpus som kunde röra sig.** Att felet ändå bara visat sig en gång, i en jämförelse som fångade det, är ett argument för att provsviten är rätt konstruerad — inte för att korpusen var det.

---

6. Implementation

nvz skrevs under provningens gång om från Python till en fristående C++17-binär (367 kB, 318 kB strippad): nvcm och nvjpg länkas in och exekveras som arbetarprocesser via `/proc/self/exe`, zstd/xz/zlib anropas som bibliotek i processen, och `liten` är standardläge. Kompressionsresultaten är ekvivalenta inom stickprovsbruset (kodekarna är samma bibliotek); skillnaden är tidsprofilen där Python deltog per fil:

mätpunkt	Python	C++
kod, läge `snabb` (5 974 filer)	2,7 s	0,8 s
gif, läge `snabb`	0,6 s	0,1 s
516 JPEG-omvandlingar	≈8 s	2,8 s
foto, uppackning (1 143 omvandlade)	7,8 s	17,4 s

Den kvarstående regressionen på foto-uppackning beror på arbetarprocessernas uppstartskostnad per bild och är den enda punkt där Python-implementationens processpool var snabbare.

Porteringen exponerade tre fel av det slag som endast uppträder under samtidighet respektive formatmigration, dokumenterade här då de är generella:

1. **Allokering mellan `fork` och `exec` i flertrådad process.** Barnet ärver låstillstånd från trådar som inte följer med; en allokering i barnet kan då blockera permanent. Symptom: arbetare med 0 % CPU i 84 minuter. Åtgärd: argumentvektorn byggs före `fork`; mellan `fork` och `exec` förekommer endast async-signal-säkra anrop.
2. **Filbeskrivararv mellan parallella arbetare.** Ett barn som skapas medan en annan tråds rör är öppna ärver även dem; motpartens läsände får då aldrig EOF. Åtgärd: `pipe2(O_CLOEXEC)`.
3. **Hashlängd i v1-registret.** Python-registret lagrar de första 16 hextecknen av objekthashen — 8 byte. En läsare som jämför 16 byte underkänner friska arkiv.
4. **Ofullständiga huvudberoenden i bygget.** Regeln för objektfilerna löd `%.o: %.cpp src/cpp/nvz.hpp` och tog alltså hänsyn till ett enda huvud. Ett fält tillagt i `struct Val` (arkiv.hpp) byggde om `huvud.o` och `arkiv.o` men inte `yta.o`, och binären länkades av objektfiler som var oense om strukturens layout. Utfallet var en binär där `nvz packa` fungerade felfritt medan `nvz -t` kraschade med SIGSEGV — ingen kompilatorvarning, ingen länkfel. Felet hittades av `prov/slumpprov.py` (avsnitt 5) på första rundan och är den tydligaste illustrationen av varför slumpprovet finns: det provar hela ytan, inte den väg man råkar använda för hand. Åtgärd: beroendena genereras av kompilatorn (`-MMD -MP`) och inkluderas i Makefile.

Kodeklagret, 2026-11-18. Sex ändringar, alla med mätvärde i 4.2, 4.5 och 5.3: LZMA:s ordbok skalas efter blocket i stället för att vara `preset 9e`:s fasta 64 MiB; ordboken blir aldrig större än blocket och krymps dessutom tills den ryms i det lediga minnet delat på antalet samtidiga kodare; `pb = 0` provas som variant för data utan ordlängd; kodekprovet görs per innehållsklass i stället för per block; den opaka tvåstegsregelns billiga steg är `zstd −3` i stället för `zstd −19`; och i läget `liten` prövas valet om på hela blocket när stickprovets försprång var tunt. Registret är version 3 (5.3). Samtliga ändringar är bakåtkompatibla vid LÄSNING — v1, v2 och v3 läses av samma binär — men ett v3-arkiv kan inte läsas av en nvz byggd före den dagen.

1. **Kontrollens gräns, och tre fel som bodde utanför den.** En kodgranskning av Ted Lyngmo 2026-11-18 hittade tre brister som varken provsviten, återskapningsproven eller slumpprovet kunde se, och de hade samma orsak: proven packar och packar upp PÅ SAMMA MASKIN, i ett lyckat flöde, med läsbara filer och ledig disk.

PNG-receptet är inte portabelt. Receptet är fyra zlib-parametrar, och antagandet att de bestämmer utströmmen håller inte — utdatan beror på implementationen, och zlib-ng ger andra bytes än zlib. Ett arkiv packat mot den ena gick inte att packa upp mot den andra. Vid åtgärden visade det sig vara värre: `png_aterskapa` kopierade skelettets CRC rakt in i utdatan, så en ström med rätt LÄNGD men fel bytes hade blivit en PNG med fel bilddata och en korrekt CRC — tyst korruption. Nu verifieras varje IDAT-bit mot sin CRC, receptet bär ett beteendebaserat avtryck av deflate-motorn (fast provsträng, crc32 på utdatan), och vid miss söks nivå, minnesnivå och strategi om; fönstret varieras inte, det står i filens eget zlib-huvud. Återskapas strömmen ändå inte vägrar nvz med båda avtrycken i felet. Kvar står att ett PNG-tungt arkiv kan behöva en maskin med samma motor — det är formatets pris och det ska stå där det syns.

Skrivfel gav tyst dataförlust. Ingen skrivning i arkivvägen kontrollerades, och gzip-ytan raderar källan efter en lyckad packning. Full disk gav alltså stympat arkiv, slutstatus 0 och raderat original. Nu kontrolleras varje `fwrite`, `fflush` och `fclose`, det `fsync`:as före stängning, och ett arkiv som inte blev färdigt tas bort. SIGXFSZ ignoreras med flit: gzip dör på 153 och hinner inte städa, nvz får EFBIG och ger 1.

En oläsbar fil dödade processen. `nvz a b c` med b oläsbar gick ur genom `main` och dog i `std::terminate` — SIGABRT, slutstatus 134, med a:s original redan raderat och c aldrig behandlad. Nu fångas varje operand för sig: varning, resten packas, slutstatus 1. Det är gzips kontrakt, mätt på samma fall.

Alla tre är regressionsprov i `prov/brister.sh`.

Filnamnen i `src/cpp/` bytte språk 2026-11-19. Texten i det här avsnittet skrevs mot de svenska namnen; mappningen är huvud→main, yta→cli, arkiv→archive, kodek→codec, omvandla→transform, trad→tree, sjalv→worker, verktyg→util. Själva koden, kommandoraden och README är numera engelska, med de svenska

kommandonamnen kvar som permanenta alias — så varje kommando i den här rapporten fungerar oförändrat.

7. Förarbete

En invändning i den offentliga diskussionen om det här arbetet förtjänar att stå i rapporten och inte bara i en kommentarstråd: *varje enskild metod nvz använder är känd och publicerad sedan länge.* Det är riktigt. Tabellen nedan ger varje del sitt förarbete, och sista kolumnen säger var nvz ligger i förhållande till det.

nvz-del	känt förarbete	nvz mot förarbetet
likhetssortering före packning	*Sort Before Compression* (<https://gwern.net/sort>); solida arkiv i RAR och 7-Zip; lrzip	samma idé. nvz sorterar på innehållsklass och ändelse, och mäter effekten separat (7.1)
deduplicering av identiska filer	borg, restic, ZFS; innehållsdefinierad chunkning sedan LBFS (2001)	svagare: nvz dedupar hela filer, inte chunkar
deflate-ström tillbaka till rådata plus recept	precomp (Christian Schneider), preflate, reflate	smalare: nvz klarar bara strömmar som zlib självt har skrivit
förlustfri JPEG-omkodning	packJPG (Matthias Stirner), Lepton (Dropbox, 2016), brunsli → JPEG XL	**sämre**: nvjpg ger −15,1 %, publicerade siffror för de andra ligger kring −20 till −22 %
kontextblandning, logistisk mixer, APM/SSE	PAQ-familjen (Matt Mahoney), lpaq, zpaq, cmix	mindre modell: paq8px slår nvcm på enwik8, 15,8 mot 20,4 MB (3.4)
innehållsmedvetet filterval per datatyp	FreeArc (Bulat Ziganshin), zpaq:s metodsträngar	samma tanke, annan mekanism: nvz väljer genom stickprov på data i stället för genom regel på filtyp
omvandling verifierad i båda riktningarna före den används	precomp och Lepton gör samma sak	samma

Siffrorna för packJPG, Lepton, brunsli och paq8px är **publicerade värden, inte mätta här**; undantaget är paq8px på enwik8, som ingår i 3.4.

Vad som återstår som eget är därmed inte en algoritm utan tre saker: att delarna sitter i ett verktyg med ett gemensamt arkivformat och en gemensam återskapningsgaranti; att valet mellan kodekar avgörs genom att prova på data i stället för att följa filändelsen; och mätningen i den här rapporten, inklusive motinstansen den hittade och rättade (4.2). nvz slår alltså inte informationsteorin, och gör ingenting ingen har gjort: den paketerar känd teknik och redovisar vad paketeringen är värd.

7.1 Hur mycket kommer från sorteringen?

Frågan går inte att besvara med ett arkiv i taget, eftersom knepen mäts ihop. `experiment/ablation.sh` håller därför **kompressorn konstant** — samma `xz -9e`, samma 64 MiB ordbok — och slår på ett knep i taget med flaggorna `--utan-sortering`, `--utan-klassning`, `--utan-dedup` och `--utan-omvandling`. Korpusen `kod` är 55 MiB och ryms helt inom ordboken i varje rad, så fönsterstorleken är inte en fri variabel: det som skiljer raderna är ordningen och ingenting annat.

Ablation, korpus `kod` (55 MiB), kodek `xz -9e` genomgående. Hela korpusen ryms i ordboken (64 MiB), så kompressorn ser varje fil från varje annan fil i varje rad:

steg	byte	mot tar	steget självt
tar i katalogordning (ingen nvz)	8 611 308	±0	—
nvz, alla knep av	8 673 944	+0,73 %	+0,73 %

steg	byte	mot tar	steget självt
+ ett block per innehållsklass	8 673 942	+0,73 %	−0,00 %
+ sortering inom klassen	8 676 765	+0,76 %	+0,03 %
+ deduplicering	8 675 644	+0,75 %	−0,01 %
+ omvandlingar	8 675 644	+0,75 %	±0

Ablation, korpus `png` (431 MiB), kodek `zstd −19 --long=27` genomgående:

steg	byte	mot tar	steget självt
tar i katalogordning (ingen nvz)	264 771 708	±0	—
nvz, alla knep av	268 949 955	+1,58 %	+1,58 %
+ ett block per innehållsklass	268 949 973	+1,58 %	±0
+ sortering inom klassen	268 949 972	+1,58 %	−0,00 %
+ deduplicering	258 596 586	−2,33 %	**−3,85 %**
+ omvandlingar (png/jpeg)	227 591 646	**−14,04 %**	**−11,99 %**

Resultatet är tydligt och delvis obekvämt. **Sorteringen mäter ±0 på båda korpusarna**, och klassindelningen likaså. Vad som mäter är omvandlingarna (−12,0 % på png) och dedupen där det finns dubbletter (−3,9 % på png, där 28,7 % av byten är dubbletter; ±0 på kod, där 0,7 % är det). Därtill kostar nvz:s eget arkivlager +0,7 till +1,6 % i de rader där ingenting är påslaget — register, blockhuvuden och att blocken packas var för sig.

Noll är emellertid inte samma sak som värdelös, och orsaken till nollan är inte sorteringen utan korpusarna: `png` har en enda filtyp, så sortering på (klass, ändelse, sökväg) ger samma ordning som katalogordningen redan hade, och `kod` ryms helt i ordboken. Ett riktigt projektträd har dessutom redan sina likheter samlade per katalog — `.py` ligger med `.py` i samma paket — så det finns ingen utspriddhet att åtgärda. Med `zstd −3` (2 MiB fönster) på samma kodkorpus mäter sorteringen till och med svagt negativt, +0,14 %: den bryter upp en lokalitet som katalogordningen redan gav.

`experiment/sortering.sh` bygger därför det fall sorteringen finns för, av exakt samma data: samma 5 974 filer ur `korpus/kod`, kopierade till 200 kataloger med slumpade namn (fast frö), så att katalogordningen förlorar all lokalitet medan innehållet är oförändrat.

ordning	`zstd −3` (2 MiB fönster)	`xz −9e` (64 MiB ordbok)
tar i katalogordning	13 926 644	8 977 616
nvz utan sortering	13 945 434	9 066 545
nvz **med** sortering	**12 833 126**	**8 869 852**
sorteringens värde	**−8,0 %**	**−2,2 %**

Där står svaret, och det ger invändningen rätt: sorteringens värde är en funktion av fönstret och av hur utspridda likheterna låg från början. Med ett litet fönster och utspridd likhet är den värd 8 %; med en ordbok som täcker hela korpusen 2,2 %; på korpusar som redan är ordnade, noll. nvz:s mätbara vinster på materialet i den här rapporten kommer alltså från omvandlingarna, från dedupen där dubbletter finns, och från nvcm i läget `liten` — **inte** från sorteringen.

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

8. Begränsningar och vidare arbete

Ordnade efter uppskattad avkastning. Posterna 1–3 och 8–10 kom ur mätningar gjorda 2026-11-18 med anledning av frågor i den offentliga diskussionen av arbetet; de bär ett mätvärde och inte bara en gissning.

1. **Minnet skalar med hela trädet, inte med blocket.** Mätt på linuxkärnan 6.12 (1,48 GiB i 86 618 filer): toppminne 2,1 GiB i läge `normal` och 3,3 GiB i `liten` — 1,4 respektive 2,2 gånger rådata. Python-implementationen strömmade varje block till en temporärfil; C++-omskrivningen håller i stället varje blocks rådata i `Bygge::ra` till dess blocket packas. Det största träd nvz kan packa är därmed ungefär tillgängligt RAM delat med 1,4, och för PNG-tunga träd mindre än så, eftersom en utvecklad PNG får vara upp till 150 gånger sin filstorlek (`PNG_UPPBLASNING_TAK`). Åtgärd: låt `Bygge::ra` ligga i en temporärfil som skrivs igenom och läses tillbaka strömmande i steg 5. Det tar bort verktygets hårdaste gräns och är den enda posten på listan som påverkar vad nvz alls klarar.

1. **Standardläget är det mest extrema.** `liten` är standard. Mätt (3.2): där `liten` vinner storlek kostar den 50–100 gånger längre uppackning — kod 68 s mot 1 s för 12 % mindre arkiv, logg 203 s mot 2 s för 17 %, enwik8 135 s mot 2 s för 14 %, Silesia 101 s mot 2 s för 6 %. Där den inte vinner något kostar den heller ingenting: på png, foto, gif, film och arkiv är `liten` och `normal` lika stora och lika snabba, eftersom kodekprovet avvisar nvcm av sig själv. Avvägningen är alltså verklig och fungerar som tänkt; invändningen gäller bara att standardvärdet gör valet åt användaren. Åtgärd att överväga: `normal` som standard, med `liten` som uttalat val.

1. **Småfilsgolvet, och det är registret — inte kompressionen.** Ett tomt arkiv är 175 byte (gzip: 25), och under ungefär 30 kB indata är nvz större än `xz -9e`: +14,2 % vid 1 kB, +2,9 % vid 10 kB, −11,8 % vid 100 kB, −14,6 % vid 1 MiB (samma textfil, växande prefix). Uppdelningen av ett 1 kB-fall visar var byten ligger:

	byte
nvz, hela arkivet	791
varav komprimerad nyttolast	616
varav register och svans	175
`xz -9e`, hela filen	696
varav xz:s container (huvud, blockhuvud, index, svans)	≈60

nvz:s **nyttolast är alltså 80 byte mindre än xz:s hela fil** — det är registret som förlorar loppet, tre gånger xz:s container. De tjocka posterna är objekthashen (16 råa byte, och för ett enda objekt finns ingenting att deduplicera mot), tidsstämpeln som ISO-sträng (~20 byte), klass- och kodeknamn som strängar i stället för nummer (~17), samt svansen på 16 byte med två 64-bitarstal. Åtgärd: ett kompakt register för enfilsarkiv. Det kräver ett v3-format, så det är ett formatbeslut och inte en kodrättning: för att gå om xz vid 1 kB behöver registret ner till omkring 60 byte, och de uppräknade posterna ger ungefär hälften.

1. **Stickprovets blinda fläck — mätt 2026-11-18, delvis åtgärdad.** Provet tar 4 MiB som åtta lösryckta bitar ur blocket. Konstruktionen är riktig för att se vad datan *är* (2.2), men den river sönder den långdistansstruktur som skiljer kodekerna: åtta bitar ur skilda delar av blocket har inga upprepningar mellan sig. På ett 120 MiB-block ur arkivkorpusen valde provet `zstd19` och fick 100,13 MiB, medan xz med täckande ordbok gav 91,97 — **provet missade 8,2 %**.

En billig förhandssignal prövades och underkändes: `zstd -3`:s kvot på provet mot samma kvot på hela blocket skiljer sig (1,016 respektive 1,025 på två block) men rangordnar dem fel, så den kan inte ersätta en mätning. Åtgärden blev därför att i läget `liten` packa om hela blocket med xz när provet valde något annat — men bara när provets försprång var tunt (under 8 %), eftersom en ovillkorlig verifiering tog enwik8 från 131 till 259 sekunder utan att spara en byte.

Kvar står att `normal` fortfarande litar på provet, och att tröskeln på 8 % är satt av två mätpunkter. Ett prov som växer med blockstorleken är fortfarande omätt.

1. **nvjpg ligger under känd teknik.** −15,1 % mot publicerade −20 till −22 % för packJPG, Lepton och brunsli (7). Avståndet är ett modelleringsavstånd: Lepton modellerar grannkoefficienter i två dimensioner och över blockgränser, nvjpg i en. Därtill lämnas progressiv JPEG rå.

1. **PNG-täckningen slutar vid zlib.** Strömmar skrivna av zlib-ng, libdeflate eller Chrome går inte att återskapa med nvz:s receptsökning. preflate löser det allmänna fallet (7); att binda in det vidgar täckningen utan att röra formatet.

1. **GIF-omkodning** (4.4): samma konstruktion som nvjpg, tillämpad på LZW-lagret.

1. **Dedup på chunknivå i stället för filnivå.** Filnivå hittar ingenting i roterade loggar, diskavbilder och versioner av samma fil, där innehållet är förskjutet. Innehållsdefinierad chunkning är känd sedan LBFS (2001) och används av borg och restic.

1. **Blockstorleken var en kompileringskonstant** (`BLOCKTAK`, 192 MiB) och är nu flaggan `--blocktak MiB`. Antalet block styr både parallelliteten (ett block packas av en tråd, och på 24 trådar står halva maskinen stilla när ett träd ger elva block) och kvoten (ett block är kompressorns fönster). Mätt på PNG-korpusen i läge `normal`:

blocktak	packtid	arkiv	mot 192 MiB
192 MiB (standard)	261 s	218 956 236	±0
64 MiB	201 s	221 495 996	**1,30× snabbare, +1,16 %**
32 MiB	137 s	231 569 236	**1,90× snabbare, +5,76 %**

Avvägningen är alltså verklig och svagt konvex: de första stegen nedåt köper mycket tid för nästan ingen storlek, och därefter blir storleken dyr. Undantagsregeln för opaka block (4.2) står kvar och berörs inte av flaggan. Om standardvärdet ska ändras är omätt — det beror på vad man optimerar — men valet går nu att göra utan att bygga om.

1. **Portabilitet.** POSIX-beroenden i C++-delen: `fork`, `pipe2`, `/proc/self/exe` (fyra ställen), `lstat` (sju), `utimensat`, `dirent`, `symlink`. macOS kräver `_NSGetExecutablePath` samt `pipe` med `fcntl`; Windows kräver därtill att arbetarmodellen byter `fork`/`exec` mot `CreateProcess` eller trådar, och ett avgörande om symlänkar och rättigheter. I dag: Linux, eller WSL2 på Windows.

1. **nvcm på opaka block:** läget `liten` sänder endast textklasser till nvcm; huruvida opak data lönar sig är omätt.

1. **AVX-512 finns i maskinen men används inte.** nvcm:s inre slinga är en skalärprodukt över tio int16-vikter per bit plus tio tabellslag. VNNI kan ta skalärprodukten, men tabellslagen är spridda och det är de som sannolikt dominerar; posten står sist därför att vinsten är omätt och kan vara noll.

1. **Korpusen är hårdlänkad till levande arbetsträd** (5.4). Avtrycket i `matning/matning.py` upptäcker nu när den rör sig, men den rätta åtgärden är att frysa den: `python3 korpus/bygg-korpus.py --kopiera`, följt av en ommätning av hela matrisen. Till dess bär varje tal mätt före 2026-11-18 den osäkerheten.
2. **Minnesbudgeten är inte deterministisk.** Ordbokens storlek — och därmed både kvot och packtid — beror på hur mycket minne maskinen har ledigt när nvz startar (4.5). Det gör mätningar beroende av annan last. Åtgärd: sätt budgeten till en andel av maskinens TOTALA minne i stället för det lediga, och redovisa andelen i rapporten.

Utanför provningen: gits packfiler (samma receptansats som PNG, oprövad) samt GPU-läget — mätt separat med slutsatsen att LZ77-matchningen accelereras 2,5 gånger men att seriell entropikodning begränsar helheten (Amdahl; se README).

Appendix A: Reproduktion

Källkod och samtliga mätdata är publicerade på <https://github.com/hugowikstrom/nvz>. Beroenden: g++ (C++17), libzstd-dev, liblzma-dev, zlib1g-dev; för jämförelsemätningarna därtill gzip, pigz, bzip2, zstd och xz.

```
git clone https://github.com/hugowikstrom/nvz && cd nvz
make                                          # bygger nvz, nvcm, nvjpg
python3 korpus/bygg-korpus.py gif film arkiv logg     # lokala korpusar
export NVPACK_SKRAP=$PWD/.skrap
python3 matning/matning.py <korpus>                   # arkivnivå, per korpus
python3 matning/standardkorpus.py silesia enwik8      # kodeknivå, en tråd
python3 prov/aterskapning.py gif arkiv logg film png foto kod
bash prov/prova-allt.sh                               # kantfall, 44 prov
python3 prov/slumpprov.py 400                         # slumpade träd, 400 rundor
bash experiment/karntest.sh                           # linuxkärnan, md5 före/efter (3.7)
bash experiment/corpora.sh                            # zlib-ng/corpora, åtta korpusar (3.8)
bash experiment/ablation.sh kod xz9e                  # ett knep i taget (7.1)
bash experiment/ablation.sh png zstd19
bash experiment/sortering.sh                          # sorteringen mot fönsterstorleken (7.1)
bash experiment/arkiv-fonster.sh                      # hypotesprövningen i 4.2
python3 experiment/dubbletter.py
python3 rapport/bred-provning.py                      # fyller tabellerna här
python3 matning/rapport.py --in-i-readme
```

Standardkorpusarna hämtas en gång:

```
curl -sSLo /tmp/silesia.zip https://sun.aei.polsl.pl/~sdeor/corpus/silesia.zip
curl -sSLo /tmp/enwik8.zip  https://mattmahoney.net/dc/enwik8.zip
mkdir -p korpus/silesia korpus/enwik8
unzip -oq /tmp/silesia.zip -d korpus/silesia
unzip -oq /tmp/enwik8.zip  -d korpus/enwik8
```